On the Algebraic Structure of Declarative Programming Languages

Gianluca Amato[**,a], James Lipton[*,b], Robert McGrail[c]

[a] *Dipartimento di Scienze, Università degli Studi "G. d'Annunzio", viale Pindaro 42, 65127 Pescara, Italy*
[b] *Department of Mathematics, Wesleyan University, Middletown CT 06459, USA*
[c] *Reem-Kayden Center 207, 31 Campus Road, Bard College, Annandale-on-Hudson, NY 12504B, USA*

Abstract

We develop an algebraic framework, *Logic Programming Doctrines*, for the syntax, proof theory, operational semantics and model theory of Horn Clause logic programming based on indexed premonoidal categories. Our aim is to provide a uniform framework for logic programming and its extensions capable of incorporating constraints, abstract data types, features imported from other programming language paradigms and a mathematical description of the *state space* in a declarative manner. We define a new way to embed information about data into logic programming derivations by building a sketch-like description of data structures directly into an indexed category of proofs. We give an algebraic axiomatization of bottom-up semantics in this general setting, describing categorical models as fixed points of a continuous operator.

Key words: categorical logic, indexed categories, logic programming, abstract data types, constraint logic programming

1. Introduction

A large number of so-called Declarative Programming Languages has been developed over the past thirty years with one common feature. A mathematical formalism underlies the text of any program, which must be respected by its computational behavior. In fact, in most logic programming languages, the code itself is taken as an executable specification that can be read and understood independently of any notion of how the computation will proceed (so-called *declarative transparency*). The paradigm has proven remarkably successful in

[*] Principal corresponding author. Phone: +1 (860) 685-2188. Fax: +1 (860) 685-2571
[**] Corresponding author. Phone: +39 085 453-7686. Fax: +39 085 4549-755
Email addresses: amato@sci.unich.it (Gianluca Amato), jlipton@wesleyan.edu (James Lipton), mcgrail@bard.edu (Robert McGrail)

a number of domains. Perceived early limitations in both expressive power and efficiency, however, have led to widespread and increasingly sophisticated attempts to strengthen the paradigm without compromising declarative transparency.

The results have included extensions of the original Horn-clause core of Prolog [56] to higher-order logic [66, 63, 12], or to non-classical logics [65, 35, 6] with more connectives and proof rules, the incorporation of different mathematical formalisms and trusted foreign algorithms via constraints [40], the addition of abstraction [63, 61], formalized metaprogramming rules [74], state-sensitive logics [62], narrowing [33], formal theories of abstract syntax [13], etc.

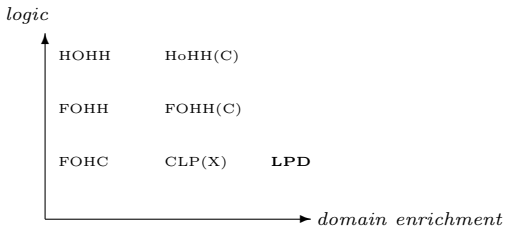
As a result of this continual evolution, declarative programming languages have seen an unprecedented amount of retooling, at the language, semantics, static analysis, code transformation, interpreter and compiler levels, and continue to do so.

The aim of the research described in this paper is to define an algebraic framework for uniform treatment of this changing field, ultimately with the hope of achieving scalability to all of these levels. The idea, some components of which have been pursued in one shape or another since 1985, is to recast syntax, semantics, state space and operational content in a uniform categorical setting capable of incorporating the various dimensions of present and foreseeable logic programming extensions. In fact categorical *syntax* can be viewed, and is viewed in this paper, as a generic language extension in its own right. Critical to this undertaking, and what sets it apart from prior work by the authors and others in the field, is the simultaneous description of syntactic, semantic, and operational layers and their interaction, as well as the inclusion of the state space as a mathematical component of the formalism.

On the underlying logic. As mentioned above, efforts to improve the efficiency and expressive power of logic programming have led to various kinds of extension to the first-order Horn Clause (FOHC) core of logic programming. One can (very) roughly divide these into *logical extensions* to the underlying syntax and proof theory, such as First-order Hereditarily Harrop (FOHH [65]), Equational (Eqlog [31]), Linear (LoLLi [35]), Modal [6] or Higher order logic (HoHC [66], HoHH [63], HiLog [12], Hiord [11]) and extensions of the syntactic *domain*, such as constraint logic programming [40], Prolog III [14], FoHH(C) and HoHH(C) [53] in which the syntax may refer to an underlying model (or associated theory) that is not the Herbrand Universe, such as the domain of the reals, finite sets, or infinite rational trees [42].

Although categorical extensions to logic programming of a more restricted nature have been proposed for richer underlying logics (e.g. FOHH) [24, 47], here we define a framework sufficiently strong to build state, data and constraint information directly into the Horn Clause syntax, and leave the extension of these methods to linear and higher-order logics for future work. Since this framework permits the definition of datatypes, monads and modules (Section 8), the need for extending the logic "along the vertical axis" (see below) may not be so critical. In keeping with this somewhat oversimplified sketch of the logic

programming extension scenario, our work on Logic Programming Doctrines (LPD) might be placed as follows:



Categorical methods. Given the wide variety of declarative extensions to the Prolog core, the fact that category theory has offered a general accounting of almost every logical system known would be reason enough to undertake a study of a categorical framework for the subject, and define (as is done here) an extensible category-based logic programming language. Yet another reason is the elimination of variables in categorical logic, and of much of the painstaking attention one must give to name clashes in conventional syntax.

Even more to the point, categorical descriptions and category-directed implementations have proven of critical utility in design, specification and modeling of other programming paradigms, especially functional programming [36, 48, 49], data type definition [73, 80, 81, 43, 38, 32, 67], object-oriented language semantics [75, 38, 30], polymorphism [5]. Notably in the case of monads [64], they have offered guidelines for safe language extensions that have been incorporated directly into the syntax of several modern programming languages (such as Haskell [69]) and into the structure of software itself. Building some of these approaches into the notion of declarative program and proof offers a promising and versatile blueprint for logic programming extensions incorporating other language paradigms.

Categorical logic programming. Most categorical extensions of logic programming start from a fundamental observation due to Goguen, Asperti and Martini [28, 4] and a number of other researchers[1] in more-or-less equivalent form:

unifiers are equalizers in a certain syntactic category.

This single fact already provides a way to capture many logic programming extensions, namely by considering a wider class of categories than the strictly syntactic ones associated with the Herbrand Universe.

We will use this observation as an entry point to categorical logic programming, gradually developing generalizations until arriving at the fundamental syntactic and semantic frameworks of LP doctrines.

[1] Burstall and Rydeheard, however, have introduced an interesting *co*-equalizer formulation in [76].

The State Space. LP doctrines are indexed premonoidal categories, to be defined below. Such categories are equipped with a *base category*, together with categories associated with each object in the base, known as the *fibers*. Goals and derivations between them will live in the fibers, functors between the fibers will play the role of generalized substitutions, and the base category will contain different formulations of the local state information, a new feature in logic programming semantics. Depending on the model in question, *state* can include the current type, data type information, the current substitution, the current constraint, current bindings. The base category may also include the current program as objects, since in Hereditarily Harrop Logic Programming [63] programs may be updated during execution. We will not explore this notion of state here, since we restrict attention to Horn Clause programming. See [24] for a categorical treatment of program update. In the examples considered in Section 8 the state space will also include a functor targeted at the base category, incorporating information about data and control.

Contents. This is the plan of the paper. In Section 2 we fix definitions and terminology which will be used through the rest of the paper. In Section 3 we give a sketch of categorical logic. In Section 4 we motivate and introduce our approach to categorical logic programming. Section 5 defines the general framework of LP doctrines and three semantics: declarative, operational and fixpoint. In Section 6 we consider in detail the important special case of Yoneda (or correct answer) semantics. Section 7 presents a treatment of constraint logic programming as an instance of the framework, and shows its soundness with respect to the standard interpretation. In Section 8 an extensive discussion of examples is given to show how the framework developed in the preceding sections can be used to build datatype and control information into the syntax of programs and their proofs. Section 9 is devoted to a comparison of our paper with related works. Finally, we conclude with a summary of our results and a discussion of future work.

2. Notation and Conventions

In this section, we fix the notation (mostly for category theory) used in the rest of the paper. A more complete treatment of the category theory background required can be found in [3, 8, 25].

A category $\mathbb{C}$ may be thought of as a collection of arrows (see e.g. [25]). Therefore we write $f \in \mathbb{C}$ when f is an arrow in $\mathbb{C}$. We denote by $|\mathbb{C}|$ the corresponding collection of objects. Given an arrow f, $\mathsf{dom}(f)$ is the *domain* (or *source*) of f and $\mathsf{cod}(f)$ the *codomain* (or *target*). For every object A, id_A is the *identity* arrow for A. We write $f : A \to B$ to denote that f is an arrow with domain A and codomain B, while $f : A \longmapsto B$ means, in addition, that f is a monic. Given $f : A \to B$ and $g : B \to G$, the *composition* of f and g is denoted by either $f \diamond g$ or fg in diagrammatic order. Given two objects A and B in $\mathbb{C}$, $Hom(A, B)$ and $\mathbb{C}(A, B)$ denote the collection of arrows with domain A and codomain B. A *functor* F from the category $\mathbb{C}$ to the category $\mathbb{D}$ is

denoted by $F : \mathbb{C} \to \mathbb{D}$, while if $F : \mathbb{C} \to \mathbb{D}$ and $G : \mathbb{C} \to \mathbb{D}$, then $\eta : F \to G$ will mean that η is a *natural transformation* from F to G. Given categories $\mathbb{C}$ and $\mathbb{D}$, we denote with $\mathbb{D}^{\mathbb{C}}$ the category of functors from $\mathbb{C}$ to $\mathbb{D}$ and their natural transformations. If $\mathbb{C}$ is category, $\mathbb{C}^{\mathrm{o}}$ is the *opposite* category of $\mathbb{C}$. If $F : \mathbb{C} \to \mathbb{D}$ is a functor, $F^{\mathrm{o}} : \mathbb{C}^{\mathrm{o}} \to \mathbb{D}^{\mathrm{o}}$ is the corresponding functor between opposite categories.

Given objects A and B in $\mathbb{C}$, we denote by $A \times B$, when it does exist, the cartesian product of A and B (which is unique up to iso), with projection morphisms given by $\pi_1^{A,B}$ and $\pi_2^{A,B}$. If $f : C \to A$ and $g : C \to B$, we denote by $\langle f, g \rangle$ the unique arrow from C to $A \times B$ such that $\langle f, g \rangle \diamond \pi_1^{A,B} = f$ and $\langle f, g \rangle \diamond \pi_2^{A,B} = g$. We will write π_1 and π_2 when A and B are clear from context. With $\mathbf{1}$ we denote the *terminator*, while $!_A$ is the only arrow from A to $\mathbf{1}$. We call *finite product category* or *FP category* a category which has all cartesian products and terminators. Most of the symbols above may optionally be annotated with the category they are referring to (like in $Hom_{\mathbb{D}}(A, B)$ or $\mathbf{1}_{\mathbb{C}}$), especially when it is not clear from the context.

A category $\mathbb{C}$ is small when the collection of its arrows is a set (instead of a proper class). It is *locally small* when $Hom(A, B)$ is a set for any pair of objects A, B. All the categories we use in this paper are locally small.

Set refers to the category of sets and total functions, and Cat the category of all small categories and functors between them. $\mathbb{N}$ and $\mathbb{R}$ denote the set of natural numbers and real numbers respectively. Given a set S, $\wp(S)$ is the power set of S, while $\wp_f(S)$ is the finite power set. Finally, if $f : A \to B$ is a function and $S \subseteq B$, then $f^{-1}(S)$ is the inverse image of S under f.

3. Categories and Logic

Categories with finite products and enough pullbacks to carry out certain required constructions can be used to give both an algebraic *syntax* and a semantics to the logic programming fragment of first order logic, following the general pattern described in [58]. The main idea is that objects, arrows and monics are the categorical counterpart of sorts, terms and predicates. Pulling back along selected arrows (terms) t of interest yields the categorical counterpart of instantiation of a predicate $p(x)$ to $p(t)$.

Compound predicates are then modeled using different categorical operations on monics depending on the level of operational fidelity to computation practice desired. For example, conjunctions of goal formulas will be captured with intersections, products and, ultimately tensor products as we strive for a more general and operationally-oriented interpretation.

From Tarski to Lawvere. This way of representing logic can easily be seen to come directly from traditional first-order Tarski semantics, appropriately presented. We will briefly outline the connection first, before proceeding with the approach developed in this paper.

So-called untyped first-order logic has sort structure given only by arities. Thus all function symbols and predicate symbols have sorts which can be denoted by natural numbers. In many sorted logic, however, function and predicate symbols may be of a specific "user-defined" sort, say `int` or `string` or products of these.

Let us consider several sorts $\sigma_1, \ldots, \sigma_n$ and ρ. In the categorical version of Tarski semantics, these are interpreted as *objects* in the category Set, i.e. by sets $[\![\sigma_i]\!]$ and $[\![\rho]\!]$. Compound sorts $\vec{\sigma} = \sigma_1 \times \cdots \times \sigma_n$ are interpreted by products of sets, i.e. $[\![\vec{\sigma}]\!] = [\![\sigma_1]\!] \times \cdots \times [\![\sigma_n]\!]$. Predicate symbols p of sort ρ are interpreted as *subsets* $[\![p]\!]$ of $[\![\rho]\!]$, which we can identify with $\{x \in [\![\rho]\!] \mid p(x)$ is true in some fixed model $\mathfrak{M}\}$. Open terms t of sort ρ with free variables $x_1, \ldots, x_n$ of sort $\sigma_1, \ldots, \sigma_n$ are interpreted as functions $[\![t]\!] : [\![\vec{\sigma}]\!] \longrightarrow [\![\rho]\!]$ in a way that is explained in detail below.

It may be best to consider an example. Suppose p is the unary predicate *even* on natural numbers. Its sort is thus `nat`. In the standard interpretation we have $[\![\texttt{nat}]\!] = \mathbb{N}$, and $[\![even]\!]$ the even members of $\mathbb{N}$. Let t be a binary function symbol on $\texttt{nat} \times \texttt{nat}$, whose output sort is `nat` and whose interpretation $[\![t]\!] : [\![\texttt{nat} \times \texttt{nat}]\!] \longrightarrow [\![\texttt{nat}]\!]$ is the right projection function π on pairs: $[\![t]\!](x, y) = \pi(x, y) = y$. The meaning of $p(t(x, y))$ should be "y is even, x is any natural number", which corresponds precisely to

$$[\![p(t(x, y))]\!] = [\![t]\!]^{-1}([\![p]\!]) = \{(x, y) \in \mathbb{N} \times \mathbb{N} \mid y \text{ is even}\} \ .$$

We have described how atomic formulas $\varphi = p(t)$ are modeled although we have left the details of how compound terms are interpreted for the following paragraph. In classical logic, Boolean combinations of formulas are interpreted using the appropriate set-theoretic operations of union, intersection and complement on the lattices $Sub([\![\sigma]\!])$ of subsets of $[\![\sigma]\!]$, for a suitable sort σ.

Finding a common sort. To interpret combinations of formulas of different sorts, we must pull their separate interpretations back (along projections) to the power set of the product of the sorts. For example, we consider how to interpret $\varphi = even(x) \wedge length3(y)$, where the sorts are: $even$: `nat`, $length3$: $\texttt{nat}^*$, and where $\texttt{nat}^*$ is the sort of strings of natural numbers with interpretation $\mathbb{N}^*$. The intended meaning $[\![length3]\!]$ of the predicate symbol $length3$ is $\{y \mid y$ is a string of natural numbers of length $> 3\}$, a member of $Sub(\mathbb{N}^*)$.

To make sense of the conjunction of the two formulas $even(x)$, $length3(y)$ we need to first pull back their separate interpretations to the power set of a common sort $\texttt{nat} \times \texttt{nat}^*$ (the product of the sorts), that is to say, take the inverse image of each along the left and right projections $\pi_1 : \mathbb{N} \times \mathbb{N}^* \to \mathbb{N}$ and $\pi_2 : \mathbb{N} \times \mathbb{N}^* \to \mathbb{N}^*$, and then take the intersection of the resulting sets, to obtain

$$\begin{aligned} [\![even(x) \wedge length3(y)]\!] &= \pi_1^{-1}([\![even]\!]) \cap \pi_2^{-1}([\![length3]\!]) \\ &= \{(x, y) \mid x \in \mathbb{N} \text{ and } x \text{ is even, and } y \text{ is a string of length } > 3\} \ , \end{aligned}$$

a subset of $\mathbb{N} \times \mathbb{N}^*$.

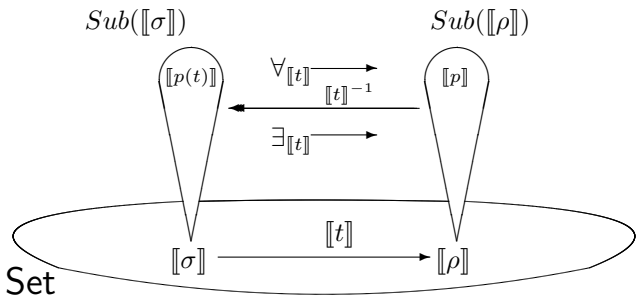


Figure 1: Interpreting atomic formulas and quantifiers in Set

Quantifiers. Let σ_1, σ_2 be sorts and π the projection of $[\![\sigma_1 \times \sigma_2]\!]$ to $[\![\sigma_2]\!]$. In the setting just given, quantification $\exists x, \forall x$ of a formula φ with two free variables x and y of sorts σ_1 and σ_2 can be captured using the operations $\exists_\pi, \forall_\pi : Sub([\![\sigma_1 \times \sigma_2]\!]) \to Sub([\![\sigma_2]\!])$ given by

$$\begin{aligned} \exists_\pi(S) &= \{b \in [\![\sigma_2]\!] \mid \exists (a,b) \in S, \pi(a,b) = b\} \ , \\ \forall_\pi(S) &= \{b \in [\![\sigma_2]\!] \mid \forall (a,b) \in S, \pi(a,b) = b\} \ . \end{aligned}$$

Thus, $[\![\exists x.\varphi]\!] = \exists_\pi [\![\varphi]\!]$ and $[\![\forall x.\varphi]\!] = \forall_\pi [\![\varphi]\!]$ yield precisely the interpretation of quantifiers given by Tarski semantics.

Lawvere [50] observed in 1969 that these operations are precisely the left and right adjoints of the inverse image $\pi^{-1} : Sub([\![\sigma_2]\!]) \to Sub([\![\sigma_1 \times \sigma_2]\!])$, or, in lattice theoretic terms, that the pairs $\langle \exists_\pi, \pi^{-1} \rangle$ and $\langle \pi^{-1}, \forall_\pi \rangle$ each form a Galois correspondence. See Figure 1 for an illustration, where π is generalized to an arbitrary arrow of the form $[\![t]\!] : [\![\sigma]\!] \to [\![\rho]\!]$, for some term t.

3.1. Logic in FP-categories

We now examine the interpretation of logic in detail in a categorical setting, sufficiently general so that both syntax and semantics will be captured by different instances of the same framework. When we describe the treatment of predicates, we will restrict attention, however, to $\exists, \wedge$ since they are the only connectives we will need for Horn Clause logic.

We start with the interpretation of terms. Assume given an FP category $\mathbb{C}$, a many sorted first order signature (Σ, Π) where Σ is a set of function symbols accompanied by their sorts, Π a set of predicate symbols and their sorts, and a set of sorted variables V. A $\mathbb{C}$-structure on (Σ, Π) is a function M that maps each sort σ to an object $M(\sigma) \in |\mathbb{C}|$ and each function symbol f of arity n, input sorts $\sigma_1, \ldots, \sigma_n$ and output sort ρ to an arrow $M(f) : M(\sigma_1) \times \cdots \times M(\sigma_n) \to M(\rho)$. Constants are functions of arity 0: for each constant c of sort ρ, we have $M(c) : \mathbf{1} \to M(\rho)$ where $\mathbf{1}$ is the terminator of $\mathbb{C}$. M also maps each predicate symbol p of sort $\vec{\sigma} = \sigma_1, \ldots, \sigma_n$ to a *monic* arrow $\longmapsto M(\vec{\sigma})$ targeted at the image of its sort. This is a natural categorical counterpart to the interpretation, in Tarski semantics, as subsets of the carrier of the sort. We sometimes abuse notation and also refer to the domain of this monic as $M(p)$ and write $M(p) \longmapsto M(\vec{\sigma})$.

A $\mathbb{C}$-structure M induces an interpretation for all open terms over V. Given a sequence $\vec{x} = x_1, \ldots, x_n$ of variables of sorts $\vec{\sigma} = \sigma_1, \ldots, \sigma_n$ respectively, we define $M(\vec{x}) = M(\vec{\sigma})$ as $M(\sigma_1) \times \cdots \times M(\sigma_n)$. Given a term t of sort ρ having all the variables among $\vec{x}$, we define an arrow $M_{\vec{x}}(t) : M(\vec{x}) \to M(\rho)$ as follows:

- $\mathbf{t = x_i}$: $M_{\vec{x}}(x_i)$ is the projection $\pi_i : M(\vec{x}) \to M(\sigma_i)$. In this case ρ is σ_i.
- $\mathbf{t = c}$: For a constant c of sort ρ, $M_{\vec{x}}(c)$ is defined as the following composition:
$$M(\vec{x}) \xrightarrow{!_{M(\vec{x})}} \mathbf{1} \xrightarrow{M(c)} M(\rho) \ . \tag{3.1}$$
- $\mathbf{t = f(t_1, \ldots, t_m)}$: If each t_i is of sort α_i, then $M_{\vec{x}}(t)$ is defined as the following composition:
$$M(\vec{x}) \xrightarrow{\langle M_{\vec{x}}(t_1), \ldots, M_{\vec{x}}(t_m)\rangle} M(\vec{\alpha}) \xrightarrow{M(f)} M(\rho) \ . \tag{3.2}$$

Given enough pullbacks, it is possible to interpret in $\mathbb{C}$ atomic formulas of first order logic. Recall that for every predicate symbol p in Π with arity n and sorts $\sigma_1, \ldots, \sigma_n$, we have a monic $M(p) \longmapsto M(\vec{\sigma})$. For an atomic formula $\phi = p(t_1, \ldots, t_m)$ with all the variables among $\vec{x}$, we define $M_{\vec{x}}(\phi)$ as the pullback of the monic $M(p) \longmapsto M(\vec{\sigma})$ along the arrow $\langle M_{\vec{x}}(t_1), \ldots, M_{\vec{x}}(t_m)\rangle$:

$$\begin{array}{ccc} M_{\vec{x}}(\phi) & \longrightarrow & M(p) \\ \downarrow & & \downarrow \\ M(\vec{x}) & \xrightarrow{\langle M_{\vec{x}}(t_1), \ldots, M_{\vec{x}}(t_m)\rangle} & M(\vec{\sigma}) \end{array}$$

The formula ϕ is considered true when $M_{\vec{x}}(\phi)$ is isomorphic to $M(\vec{x})$. In the category Set this coincides with the usual definition of truth in Tarski semantics, i.e. every member of the sort of ϕ is in its interpretation.

In the context of FP categories, substitutions and unification have a direct counterpart, too. Let $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ be an idempotent substitution and assume that all the variables in $t_1, \ldots, t_n$ are in the sequence $\vec{y}$. Then one can define a corresponding categorical substitution $\Theta_{\vec{y}}$ as the morphism:

$$M(\vec{y}) \xrightarrow{\langle M_{\vec{y}}(t_1), \ldots, M_{\vec{y}}(t_n)\rangle} M(\vec{x}) \ . \tag{3.3}$$

It is easy to prove by structural induction [22] that, given a term s with all the variables among $\vec{x}$, $M_{\vec{y}}(s\theta) = \Theta_{\vec{y}} \diamond M_{\vec{x}}(s)$:

$$M(\vec{y}) \xrightarrow{\langle M_{\vec{y}}(t_1), \ldots, M_{\vec{y}}(t_n)\rangle} M(\vec{x}) \xrightarrow{M_{\vec{x}}(s)} M(\rho) \ . \tag{3.4}$$

Application of the substitution θ above to a predicate ϕ whose sort is $M(\vec{x})$ is accomplished by taking the pullback of the monic interpreting ϕ along the arrow $\Theta_{\vec{y}}$ just defined.

Given two terms s and t of the same sort ρ with all the variables in $\vec{x}$, if θ is a unifier then $\Theta_{\vec{y}}$ equalizes $M_{\vec{x}}(s)$ and $M_{\vec{x}}(t)$, i.e., makes the following diagram commute:

$$M(\vec{y}) \xrightarrow{\Theta_{\vec{y}}} M(\vec{x}) \underset{M_{\vec{x}}(t)}{\overset{M_{\vec{x}}(s)}{\rightrightarrows}} M(\rho) \ .$$

If $\mathbb{C}$ is the pure Lawvere Algebraic Theory [52] (see also Example 5.2 below) for Σ, and θ is a most general unifier, then $\Theta_{\vec{x}}$ is an equalizer, that is to say, terminal among all equalizing arrows like the one in the preceding diagram.

Given this interpretation of classical first order languages, the successive abstraction step is *considering the FP category itself* as the language, without relying on any interpreted syntactic objects. This is common in categorical logic and it has the great advantage of allowing us to work with syntax without having to worry about variables and consequent name clashes. We will talk therefore of objects as sorts, arrows as terms or substitutions, equalizers as most general unifiers, monics as predicates, and pullbacks of predicates along terms, if they exist, as the categorical counterpart of instantiation $p(t)$ of predicate symbols p.

The doctrinal approach. We move to yet a more general categorical interpretation of logic, *indexed categories,* inspired by the diagram in Figure 1, first introduced by Lawvere [50, 51] and often used since, see e.g. [78]. This diagram, often called a doctrinal diagram when certain conditions are added, will become the defining algebraic framework for our general categorical logic.

Recall that we have interpreted predicates of sort σ as monic arrows targeted at $[\![\sigma]\!]$. But these correspond to *objects* in the subobject poset $Sub([\![\sigma]\!])$ (which correspond to subsets of $[\![\sigma]\!]$ when $\mathbb{C} = \mathsf{Set}$). Thus, the natural generalization to arbitrary indexed categories is to let a goal of sort σ be any object in the fiber over $[\![\sigma]\!]$ of a $\mathbb{C}$-indexed category [34, 45].

3.2. Indexed Categories

Indexed categories and the related notion of *fibration* have been used extensively in categorical logic. Informally, indexed categories are collections of categories ("fibers") parametrized by another so-called base category in a way that respects some of the structure of the base. They give a natural way of passing from propositional to predicate logic. We take a given signature for predicate logic as a base category. We then associate to each object (type or arity) in the signature a *fiber* consisting of (a categorical model of) a propositional logic, such as a lattice of predicates. Substitution and quantification are then captured by certain functors between these propositional fibers, directly generalizing the basic framework indicated at the beginning of this section with Tarski semantics in Set. We will use variants of this idea to give a general notion of syntax and semantics for logic programming.

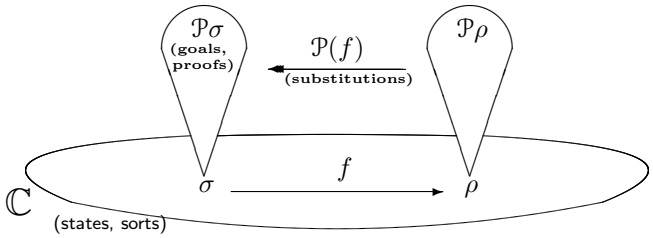
The most natural way to capture this notion of a category varying over a base category $\mathbb{C}$ is to define it as a functor (usually contravariant) from $\mathbb{C}$ to

the category Cat of all small categories and functors. That is to say we view a family of categories $\{\mathbb{F}_\sigma : \sigma \in |\mathbb{C}|\}$ indexed over the objects of $\mathbb{C}$ as a functor $\mathcal{P}$ taking each σ to $\mathbb{F}_\sigma$. Since $\mathcal{P}$ is a functor it will also map arrows between sorts to functors between the fibers, respecting composition. Note that we use greek letters for the objects of the base category $\mathbb{C}$, since they often correspond to sorts in standard logic.

We now give the definitions. A broad treatment of these topics can be found in [37].

Definition 3.1 *A* strict indexed category *over a* base category $\mathbb{C}$ *is a functor* $\mathcal{P} : \mathbb{C}^{\mathrm{o}} \to \mathsf{Cat}$. *For each object* σ *of* $\mathbb{C}$*, the category* $\mathcal{P}(\sigma)$ *is called the* fiber *over* σ*. If* f *is an arrow in the base category,* $\mathcal{P}(f)$ *is called a* reindexing functor*. We will write* $f^\sharp$ *instead of* $\mathcal{P}(f)$ *when this does not cause ambiguities.*

The following diagram illustrates the fundamental components of an indexed category:



For future reference, we have also indicated the logic programming concepts (**goals & proofs, substitutions, states & sorts**) formalized by these components.

Indexed Functors and Change of Base. Given indexed categories $\mathcal{P}$ and $\mathcal{Q}$ over $\mathbb{C}$ and $\mathbb{D}$ respectively, an *indexed functor* from $\mathcal{P}$ to $\mathcal{Q}$ is given by a *change of base* functor $F : \mathbb{C} \to \mathbb{D}$ and a natural transformation $\tau : \mathcal{P} \to (F^{\mathrm{o}} \diamond \mathcal{Q})$. In the following, when we have an indexed functor $H = \langle F, \tau \rangle : \mathcal{P} \to \mathcal{Q}$, we will often use $f^\sharp$ for $\mathcal{P}(f)$ and $f^\flat$ for $\mathcal{Q}(Ff)$.

In a *non-strict* indexed category, $\mathcal{P}$ is only required to be a pseudo-functor $\mathbb{C}^{\mathrm{o}} \to \mathsf{Cat}$, meaning that $\mathcal{P}$ need only preserve identity and composition of arrows up to isomorphism. This generality is often forced on us. Just consider the following case, sometimes taken to be the defining example of the subject. Take reindexing functors to be pullbacks between subobject lattices, which only preserve composition up to iso. We will derive our indexed categories by other means, which automatically give us strict indexed categories. Therefore, in the following we will omit the prefix "strict".

Choosing the right operations in the fibers. The algebraic structure of the fibers of our indexed categories will determine how free we are to build extra information into the definition of predicates, and how operational we make our categorical syntax. To this end, instead of endowing fibers with finite products

as in [78] and some prior work by the authors and collaborators [24, 55] we choose to model conjunction of goals with monoidal structures below.

Our categorical syntax must allow us to represent code and execution directly, that is to say, programs, goals and sequences of proof steps. Goals will be objects in the fibers, and sequences of proof steps will be arrows. Since backchaining involves moving from theorems to premises, these arrows will initially be in the opposite direction of backchaining.

If we choose to model conjunctions of goals $\mathbf{G}_1, \mathbf{G}_2$ using products, we are forced to allow projections such as $\mathbf{G}_1 \times \mathbf{G}_2 \to \mathbf{G}_1$, which in turn forces us to allow (in reverse) resolution steps of the form

$$\mathbf{G}_1 \rightsquigarrow \mathbf{G}_1, \mathbf{G}_2.$$

In conventional categorical proof theory, this is just a form of weakening, hence sound from a purely logical point of view. In fact this proof step does not spoil soundness or completeness of Prolog, since a successful derivation $\mathbf{G} \rightsquigarrow \cdots \Box$ exists using this rule if an only if it exists without this rule (by induction on the length of proof). But it is clearly an absurd step from a logic programming point of view.

Modeling conjunction with intersection (of goals interpreted as sets) is even worse *in the syntax.* It forces us to identify goals such as $\mathbf{G}_1, \mathbf{G}_2$ with $\mathbf{G}_2, \mathbf{G}_1$, and with, for example, $\mathbf{G}_1, \mathbf{G}_2, \mathbf{G}_1$, which can have such different observable behavior in terms of computational effects or resource consumption.

The choice of a non-commutative tensor operation makes it possible to build models in which pairs of goals such as $\mathbf{G}_1$ and $\mathbf{G}_1 \otimes \mathbf{G}_1$ or $\mathbf{G}_1 \otimes \mathbf{G}_2$ and $\mathbf{G}_2 \otimes \mathbf{G}_1$ are not even isomorphic, and in which no arrows exist between $\mathbf{G}_1 \otimes \mathbf{G}_2$ and $\mathbf{G}_1$, in general.

This use of monoidal structures is consistent with the analysis of logic programs through linear logic in e.g. [44] although we will not further pursue this connection here. We will also need to pass to the more general *premonoidal structures*, since they allow us to rule out the "parallel" resolutions of [16] (where all goals may be reduced simultaneously) in favor of standard non-deterministic one-goal-at-a-time resolution, as further discussed below.

3.3. Monoidal and Premonoidal Categories

A *monoidal category* is a category endowed with a formal product operation on objects and arrows often denoted $\otimes$, with some associativity properties. They arise naturally in algebra and linear logic semantics. For a detailed discussion and historical motivation the reader should consult [57].

Their main use here will be to formalize concatenation of goals and certain kinds of proof steps between them. In order to rule out proof steps that are not computationally meaningful we will need to consider some variants and generalizations defined below.

Definition 3.2 *A* strict monoidal category *is a category* $\mathbb{C}$ *together with a functor* $\otimes : \mathbb{C} \times \mathbb{C} \to \mathbb{C}$*, called* tensor product*, and an* identity *element* $\top \in |\mathbb{C}|$

such that, given arrows f, g and h, the following identities hold: $f \otimes (g \otimes h) = (f \otimes g) \otimes h$, $f \otimes \top = f$, $\top \otimes f = f$.

By restricting the properties of the tensor product we may obtain a *premonoidal category*, which has been introduced in [71] as a tool for analyzing side effects in the denotational semantics of programming languages. It is used here to control which transitions between goals count as legal resolution steps.

It is easier to present premonoidal categories as a particular case of binoidal categories. We remind the reader that if $\mathbb{C}$ is a category then $|\mathbb{C}|$, the class of objects of $\mathbb{C}$, can be viewed as a discrete category: the only arrows are the identities.

Definition 3.3 *A* binoidal category *is a category $\mathbb{C}$ with a pair of functors $\otimes_l : \mathbb{C} \times |\mathbb{C}| \to \mathbb{C}$ and $\otimes_r : |\mathbb{C}| \times \mathbb{C} \to \mathbb{C}$ such that $id_A \otimes_l B = A \otimes_r id_B$ for each pair of objects in $\mathbb{C}$.*

The main interest for logic programming applications is to ensure that the tensor product *can only act on one coordinate at a time*. Arrows between objects in the free binoidal category will be of the form $A \otimes_l B \to A' \otimes_l B$, which corresponds, in logic programming to selecting the goal A' (resolutions are in the reverse direction) and reducing it to A while leaving B fixed.

Since there are no ambiguities, we can denote both functors with $\otimes$, writing things such as $A \otimes B$, $A \otimes g$ or $f \otimes B$. In general, it does not make sense to write $f \otimes g$. A morhpism $f : A \to A'$ in a binoidal category is called *central* if, for every morphism $g : B \to B'$, the following equalities hold: $(g \otimes A) \diamond (B' \otimes f) = (B \otimes f) \diamond (g \otimes A')$ and $(A \otimes g) \diamond (f \otimes B') = (f \otimes B) \diamond (A' \otimes g)$. In this case, we may use the notations $f \otimes g$ or $g \otimes f$.

Definition 3.4 *A* strict premonoidal category *is a binoidal category $\mathbb{C}$ with an identity element $\top \in |\mathbb{C}|$ such that, for every $f : C \to C'$, the following identities hold: $\top \otimes f = f$, $f \otimes \top = f$, $f \otimes (A \otimes B) = (f \otimes A) \otimes B$, $A \otimes (f \otimes B) = (A \otimes f) \otimes B$ and $A \otimes (B \otimes f) = (A \otimes B) \otimes f$.*

Observe that if $\mathbb{C}$ is a strict premonoidal category where all morphisms are central, then $\mathbb{C}$ is a strict monoidal category.

In the following, we will never use the non-strict variants of monoidal and premonoidal categories, hence we will also omit the prefix "strict". Given two premonoidal categories $\mathbb{C}$ and $\mathbb{D}$, a *premonoidal functor* $F : \mathbb{C} \to \mathbb{D}$ is a functor which sends central maps to central maps and preserves identity element and tensor product on the nose. Finally, given two premonoidal functors $F, G : \mathbb{C} \to \mathbb{D}$, a premonoidal natural transformation $\eta : F \to G$ is a natural transformation such that all the arrows η_A are central, $\eta_{A \otimes_{\mathbb{C}} B} = \eta_A \otimes_{\mathbb{D}} \eta_B$ and $\eta_{\top_{\mathbb{C}}} = id_{\top_{\mathbb{D}}}$.

3.4. On some Standard Categorical Constructions

A number of standard definitions and results in category theory are repeatedly used in this paper. They are noted here for future reference. The reader should consult e.g. [57, 49, 25] for the details.

Freely generated objects and universal mapping properties. An object A in a category $\mathbb{C}$ is *initial* or *free* if for any object B in $\mathbb{C}$ there is a unique arrow from A to B. It is terminal if the dual property holds (the arrow goes the other way).

Such objects are easily seen unique up to isomorphism. We can relativize these definitions, thus obtaining (from terminal objects, for example) such notions as pullbacks, products and limits. For example, a product in $\mathbb{C}$ of two objects A and B in $\mathbb{C}$ is a span $A \longleftarrow C \longrightarrow B$ such that for any other span $A \longleftarrow C' \longrightarrow B$ there is a unique arrow from C' to C making the resulting diagram commute. We can define a category $Sp(\mathbb{C}, A, B)$ of spans into A and B in $\mathbb{C}$. Then the product of A and B is simply a terminal object in $Sp(\mathbb{C}, A, B)$.

Given two categories $\mathbb{S}$ and $\mathbb{C}$, and a functor $U : \mathbb{C} \to \mathbb{S}$, we say that an object A of $\mathbb{C}$ is freely generated by the object X in $\mathbb{S}$ if there is an arrow $X \to U(A)$ in $\mathbb{S}$ such that for every B in $\mathbb{C}$ and every arrow $X \to U(B)$ there is a unique arrow $\varphi : A \dashrightarrow B$ such that the following diagram commutes

$$\begin{array}{ccc} X & \longrightarrow & U(A) \\ & \searrow & \vdots \\ & & U(B) \end{array} \qquad U(\varphi)$$

Often there is a functor $F : \mathbb{S} \to \mathbb{C}$ which associates to each object of $\mathbb{S}$ a freely generated object of $\mathbb{C}$. In this case F and U form a pair of adjoint functors [57, 49, 25]. Examples are the free group generated by a set and the free algebra (over a signature) generated by a set of variables. Of particular interest is the *free category* generated by a graph. Taking the functor $U : \mathsf{Cat} \to \mathsf{Graph}$ to be the one that maps every (small) category to its underlying graph, one shows the existence of F by constructing, for any graph G the desired category $F(G)$ with the same objects as G, and with arrows from, say, A to B in $F(G)$ made up of sequences of arcs in G connecting A to B. See [57, 49] for details.

Using this construction one may *freely adjoin* an arrow, a collection of arrows, or a diagram (between existing objects) to a category $\mathbb{C}$, by first adding the arrows as arcs to $U(\mathbb{C})$, then constructing the category freely generated by this augmented graph, and then taking a suitable quotient to identify arrows that were equal in the original category. Freely adding a single arrow to a category is discussed in detail, from several perspectives (Kleisli categories, slices) in [49, 25].

The Yoneda embedding. We can define the functor

$$Y : \mathbb{C} \to \mathsf{Set}^{\mathbb{C}^o} \tag{3.5}$$

from any category $\mathbb{C}$ into the category of contravariant functors from $\mathbb{C}$ to Set, by mapping each object A to the functor $Hom_{\mathbb{C}}(\_, A)$ and each arrow $f : A \to B$ to the natural transformation $\hat{f} : Hom_{\mathbb{C}}(\_, A) \to Hom_{\mathbb{C}}(\_, B)$ that acts as follows: for each object X in A, $\hat{f}_X : Hom_{\mathbb{C}}(X, A) \to Hom_{\mathbb{C}}(X, B)$ sends $\alpha : X \to A$ to $\alpha f : X \to B$. An analysis of this important functor Y shows that every category

$\mathbb{C}$ can be fully and faithfully embedded in the presheaf topos $\mathsf{Set}^{\mathbb{C}^o}$, in such a way that the images of the objects in $\mathbb{C}$, called the *representable* functors in $\mathsf{Set}^{\mathbb{C}^o}$, are indecomposable projectives. Furthermore every object of the presheaf category is a colimit of representables. These results and the associated Yoneda Lemma, are presented succinctly in [49, 57]. They play a central role in this paper, since a large class of logic programming models considered here are constructed by first taking the Yoneda embedding of a syntactic category $\mathbb{C}$ into its associated presheaf category, and then extending the mapping from predicates, i.e. fibers over $\mathbb{C}$, to subfunctors of the representables.

4. Logic Programming with Categories

Categorical approaches to logic programming appeared with the categorical treatment of unification given by Rydeheard and Burstall in [76]. Asperti and Martini [4] formalize the syntax of conventional Horn clause logic using categorical tools, and a topos-theoretic semantics is given. Corradini and Asperti in [16], following some basic ideas already developed by Corradini and Montanari in [17], give a categorical analysis of logic program transitions and models using indexed monoidal categories (where the base category is always the natural numbers). Kinoshita and Power [45] give a fibrational semantics for logic programs which is very similar to [16], but more general since the algebra of terms is not assumed to be freely generated.

Finkelstein, Freyd and Lipton [24] propose a different framework for the syntax and semantics of logic programs. They do not use indexed categories, but freely adjoin generic predicates to the category $\mathbb{C}$ of sorts and terms, adhering to the interpretation of logic in finite product categories we introduced in Section 3.1. We think that indexed categories are easier to deal with, since they separate the logic part (in the fibers) from the functional part (in the base) of the languages. On the other hand, [16] and [45] focus on the operational and model theoretic side of the matter. However they lack any bottom-up denotational semantics such as that provided by the T_P operator of van Emden and Kowalski [79]. This immediate consequence operator and the characterization of models in terms of its fixed points seems to be a cornerstone of logic programming, since it appears, in one form or another, across most semantic treatments of logic programs [7, 15]. The bottom up description of program models is indispensable for recent work in static analysis and abstract interpretation and its application to compilation, built into every Prolog system in use today. For these reasons, the categorical framework in [24] includes an analogue of the T_P operator, a definition of minimal model as its least fixed point, and its bottom-up representation. Instead of using a term model, it uses the Yoneda embedding of the signature category $\mathbb{C}$ into the presheaf category $\mathsf{Set}^{\mathbb{C}^o}$, and interprets goals as subfunctors of their types in this category, as in Section 3.4. This roughly corresponds to the semantics of correct answers.

The first contribution of our paper is the integration of the indexed framework in [16, 45] with the fixed point semantics in [24]. At the same time, the two frameworks are significantly extended to include:

- The use of premonoidal structures to model conjunctions of goals instead of monoidal [16] or finite product [45, 24] structures. This gives a better operational fidelity, as already discussed in Section 3.2.

- A more general syntax which admits non-freely generated goals (i.e. two goals $p(t_1)$ and $p(t_2)$ for $t_1 \neq t_2$ may be equal, or may be related by some built-in proof). In turn, this allows:

  - The incorporation of data structures, along the lines of Lipton and McGrail work [55]. However, we argue that our presentation in terms of premonoidal indexed categories and display structures is at the same time simpler than the original work and better, thanks to the greater operational fidelity.
  - The incorporation of constraint logic programming, which is proved to be sound w.r.t. the standard treatments in [40, 53].

- A more general fixed point semantic operator, allowing the replacement of the presheaf category $\mathsf{Set}^{\mathbb{C}^o}$ with other semantic domains (such as those for ground answers, correct partial answers, etc.).

For a detailed comparison of our approach with [16, 45, 55, 24] and other papers, the reader may consult Section 9.

A different point of view is adopted by Diaconescu [21], which gives a categorical semantics of *equational logic programming* using an approach derived from the the *theory of institutions* [29]. This allows to consider equational logic programming over non-conventional structures, recovering both standard logic programming and constraint logic programming.

4.1. First Example

We will begin by rephrasing logic programming in terms of indexed monoidal categories, also called IMCs. In later sections, after we present the general theory, we will build on this framework to add data type information and constraints.

Towards indexed categories of goals and proofs. To build the indexed category we want for logic programming, we start with a base category $\mathbb{C}$ with finite products defining local logical state. Initially this will just mean types and terms (a categorical signature). The *fiber* over each object will be a category of goals.

As seen in the preceding section, each arrow in the base category induces a *reindexing* functor between the fibers, generalizing the notion of substitution, which may now include arbitrary state transitions. Unification will now just mean applying reindexing to program and goal that identify the head of a clause with the goal in question.

The display category $\mathbb{J}$. The base category $\mathbb{C}$ supplies the signature for logic programming, but information about the predicate symbols that may appear in programs, goals and proofs must be given in a so-called *display category* $\mathbb{J}$ together with a functor $\delta : \mathbb{J} \to \mathbb{C}$ that maps each predicate name to its associated sort.

We can think of $\mathbb{J}$ as a categorical analogue of the list of predicate letters in a first order language, containing the predicate names, with δ providing their type or sort assignment. In this initial example, $\mathbb{J}$ is just a discrete category.

Thus if program P, for example, has predicate symbols p_1, p_2, p_3 of types `nat`, `nat` and `nat x nat`, then $\mathbb{J}$ is the discrete category $\{p_1, p_2, p_3\}$, with only identity arrows. Assuming $\mathbb{C} = \mathsf{Set}$, then $\delta : \mathbb{J} \to \mathbb{C}$ may be given by $\delta(p_1) = \mathbb{N}, \delta(p_2) = \mathbb{N}, \delta(p_3) = \mathbb{N} \times \mathbb{N}$.

In Section 8 we will consider non-discrete categories $\mathbb{J}$ where there may be arrows other than the identities. We will also add diagrams and cones to the display category $\mathbb{J}$ allowing us to build-in new relations, modules and data types to the Horn Clause framework.

The Indexed Monoidal Category of goals. We now build the free IMC $\mathcal{P}_{\mathbb{J}}$ of *goals* generated by the predicate symbols designated by $\mathbb{J}$. The basic idea is to represent a goal $p(t)$ as a tagged arrow, that is to say a pair (p, t) where the predicate symbol p is an object in $\mathbb{J}$ and t is an arrow in $\mathbb{C}$ whose target is the sort associated with p. Compound goals $p_1(t_1), \ldots, p_n(t_n)$ will be sequences of these tagged arrows.

We now formally define $\mathcal{P}_{\mathbb{J}} : \mathbb{C}^{\mathrm{o}} \to \mathsf{Cat}$ to be the functor which maps each object σ of $\mathbb{C}$ to the discrete category whose objects are sequences of pairs (p, t) where $p \in |\mathbb{J}|$ and $t : \sigma \to \delta(p)$. If p and p' are different objects in $\mathbb{J}$ mapped to the same object of $\mathbb{C}$, that is to say $\delta(p) = \delta(p')$, the two objects $(p, t), (p', t)$ are well-defined and *distinct* in $\mathcal{P}_{\mathbb{J}}(\sigma)$. We will write the goal (p, t) also as $p(t)$ when we want to stress the analogy with standard logic programming. $\mathcal{P}_{\mathbb{J}}$ is endowed with a simple monoidal structure: the tensor $\otimes_\sigma$ is given by concatenation of sequences, while $\top_\sigma$ is the empty sequence.

We must now define the action of $\mathcal{P}$ on arrows $r : \rho \to \sigma$ in $\mathbb{C}$, which must be to produce reindexing functors between the fibers. It acts by composition: $\mathcal{P}(r) : \mathcal{P}(\sigma) \to \mathcal{P}(\rho)$ maps the goal

$$\mathbf{G} = (p_1, t_1), \ldots, (p_n, t_n) \in \mathcal{P}(\sigma)$$

to

$$\mathbf{G}' = (p_1, rt_1), \ldots, (p_n, rt_n) \in \mathcal{P}(\rho) \ ,$$

and the identity $id_{\mathbf{G}}$ to $id_{\mathbf{G}'}$.

On the sorts of goals, clauses and programs. We say that a goal $\mathbf{G}$ has sort σ when $\mathbf{G}$ is an object in $\mathcal{P}\sigma$. Consider an atomic goal $p(t)$, where $t : \sigma \to \delta(p)$ is an arrow in the base category $\mathbb{C}$. The sort of $p(t)$ is the source of the arrow t, i.e. σ. By convention, we also say that the sort of the predicate letter p is $\delta(p)$, i.e. the sort of the goal $p(id)$.

A goal $p_1(t_1), \ldots, p_n(t_n)$ can only exist if each $p_i(t_i)$ has the same sort α, since the monoidal operation $\otimes$ used to combine goals is only defined within each fiber. In this case the entire goal has sort α.

The fact that in the case of the compound goal just cited all the literals $p_i(t_i)$ have the same sort is not limiting in any way, since any attempt to construct a goal $p_1(t_1), p_2(t_2)$ where $p_1(t_1)$ is of sort σ_1 and $p_2(t_2)$ is of sort σ_2 can be done by reindexing both goals to a common fiber, namely the one over $\sigma_1 \times \sigma_2$, and taking ${\pi_1}^\sharp(p_1(t_1)) \otimes {\pi_2}^\sharp(p_2(t_2))$, where the $\pi_i : \sigma_1 \times \sigma_2 \to \sigma_i$ are projections.

Categorical derivations. Given the IMC $\mathcal{P}_\mathbb{J}$, we define a clause cl to be a pair

$$\langle p(t), p_1(t_1) \otimes \cdots \otimes p_n(t_n) \rangle$$

of goals in the same fiber $\mathcal{P}(\sigma)$, whose first component is an atomic goal (i.e., it is a sequence of length one). The object σ in the base category is the *sort of the clause*, which we prefer to write as follows:

$$p(t) \xleftarrow{cl} p_1(t_1), p_2(t_2), \ldots, p_n(t_n) \ . \tag{4.1}$$

In Section 5.1 a more general definition of clause is given.

A notion of resolution may be introduced, in analogy with [24]. Given a clause cl like (4.1) of sort $\rho \in |\mathbb{C}|$ and a goal $\mathbf{G}_1$ of sort $\sigma \in |\mathbb{C}|$, we have a resolution step

$$\mathbf{G}_1 \xrightarrow{r,s,i,cl} \mathbf{G}_2$$

when

1. $\mathbf{G}_1 = q_1(t'_1), \ldots, q_i(t'_i), \ldots, q_l(t'_l)$,
2. For some $\alpha \in |\mathbb{C}|$, $r : \alpha \to \sigma$, $s : \alpha \to \rho$ and $r^\sharp q_i(t'_i) = s^\sharp p(t)$,
3. $\mathbf{G}_2 = r^\sharp q_1(t'_1), \ldots, r^\sharp q_{i-1}(t'_{i-1}), s^\sharp p_1(t_1), \ldots, s^\sharp p_n(t_n), r^\sharp q_{i+1}(t'_{i+1}) \ldots, r^\sharp q_l(t'_l)$.

In the case of $\mathcal{P}_\mathbb{J}$, the definition above gives rise to the same derivations of [24]. However, there is the potential for a greater generality, since $r^\sharp$ and $s^\sharp$ now depend on the definition of the reindexing functors in the category of goals. This extra generality will allow us, later in the paper, to treat constraint logic programming languages in a natural way.

Given a derivation $d = \mathbf{G}_1 \xrightarrow{r_1,s_1,i_1,cl_1} \cdots \xrightarrow{r_m,s_m,i_m,cl_m} \mathbf{G}_{m+1}$, the *answer* of d is defined as the composition $r_m \diamond \cdots \diamond r_1$. A *successful derivation* is a derivation which ends with the empty goal. A *correct answer* is the answer corresponding to a successful derivation.

The Indexed Premonoidal Category of proofs. The resolution step is the fundamental notion of computation in logic programming, but we do not wish to lose sight of the fact that it is a highly controlled form of proof search, and that proofs and proof construction underlie the whole discipline. Thus, we take arrows to denote proofs, which run in the opposite direction to resolution.

A category of proofs $\mathcal{F}_P$ may be obtained by freely adding clauses in P to the corresponding fibers in $\mathcal{P}_\mathbb{J}$. This means that $\mathcal{F}_P(\sigma)$ will contain, together

with all the arrows in $\mathcal{P}_{\mathbb{J}}(\sigma)$, also arrows $p(t) \xleftarrow{cl} p_1(t_1), p_2(t_2), \ldots, p_n(t_n)$ for each clause cl of sort σ. In turn, this will generate many new proofs, given by formal reindexing of cl along arrows in the base category and by closure w.r.t. the premonoidal structure.

Actually, the extended notion of resolution defined above is also bound to an extended categorical notion of proof. A derivation $\mathbf{G}_1 \rightsquigarrow^* \mathbf{G}_2$ logically corresponds to a proof of $\mathbf{G}_1$ from $\mathbf{G}_2$. In order to expose this correspondence more clearly we restrict our focus to *flat* derivations, i.e. derivations where in each step $\mathbf{G}_1 \xrightarrow{r,s,i,cl} \mathbf{G}_2$ we only allow $r = id_\sigma$, with σ the type of $\mathbf{G}_1$. Once we fix a sort σ, we may define a proof-theoretic category $\mathcal{F}_P(\sigma)$ whose objects are goals of sort σ and arrows $\mathbf{G}_1 \leftarrow \mathbf{G}_2$ are derivations from $\mathbf{G}_1$ to $\mathbf{G}_2$. Composition of arrows is given by concatenation of derivations, while the identity is the empty derivation. Moreover, we may turn $\mathcal{F}_P$ into an indexed category. If $t : \rho \to \sigma$ and d is a derivation of type σ, then $\mathcal{F}_P(t)(d)$ is the derivation obtained by replacing in d every step $\mathbf{G}_1 \xrightarrow{id_\sigma,s,i,cl} \mathbf{G}_2$ with $t^\sharp\mathbf{G}_1 \xrightarrow{id_\rho,ts,i,cl} t^\sharp\mathbf{G}_2$.

In Section 5 we will show that $\mathcal{F}_P$ is an indexed premonoidal category (see also Example 5.16). It is the free premonoidal category obtained by adding to $\mathcal{P}_{\mathbb{J}}$ new arrows which correspond to the clauses in P.

Example 4.1 Let the base category $\mathbb{C}$ be the Lawvere Algebraic Theory for the natural numbers $\mathbb{N}$. That is to say, we take as objects a copy of the natural numbers $\{n_k | k \in \mathbb{N}\}$ representing powers of $\mathbb{N}$, i.e. arities, (but *not* representing the natural numbers), and as arrows, the closure under composition of all the countably many constants $n_0 \xrightarrow{0,1,2,\ldots} n_1$, the arrows $+, \times : n_2 \to n_1$, as well as all arrows imposed by the finite product structure. We then take the quotient of this category with respect to the equational theory of $+$ and $\times$, so that, e.g. the arrows $+\langle 1, 3\rangle$ and $\times\langle 2, 2\rangle$ will be identified. We consider the following program

```
fact(0,1).
fact(X+1,(X+1)*Y) :- fact(X,Y).
```

The first clause has type n_0 and the second one type n_2 (that is to say, $\mathbb{N}\times\mathbb{N}$) because of the two free variables. The variables X, Y are represented by the projections $n_2 \xrightarrow{\mathsf{l}} n_1$ and $n_2 \xrightarrow{\mathsf{r}} n_1$, and the pair $t = (X + 1, (X + 1) * Y)$ by

$$M_{\{X,Y\}}(t) = n_2 \xrightarrow{\langle +\langle \mathsf{l}, !_{n_1} 1\rangle, \times\langle +\langle \mathsf{l}, !_{n_1} 1\rangle, \mathsf{r}\rangle\rangle} n_2 \ .$$

Recall that e.g $!_k 3$ means, for any object k, the composition $k \xrightarrow{!_k} n_0 \xrightarrow{3} n_1$. Below, to simplify notation we will drop the $!_k$ and leave implicit the source and target of identity arrows.

Since we have only one predicate symbol *fact* of type n_2, the display category $\mathbb{J}$ will just be the set $\{fact\}$, with $\delta(fact) = n_2$. The indexed monoidal category $\mathcal{P}_{\mathbb{J}}$ has, in the fiber $\mathcal{P}_{\mathbb{J}}(n_2)$ all sequences of pairs $(fact, t)$ where $n_2 \xrightarrow{t} n_2$, representing $fact(t)$. The predicate symbol *fact* itself is represented by $(fact, id_{n_2})$.

The program clauses. are given by

$$fact(\langle 0,1\rangle) \xleftarrow{cl1} \top \ ,$$
$$fact(\langle +\langle \mathsf{I}, !_{n_1} 1\rangle, \times\langle +\langle \mathsf{I}, !_{n_1} 1\rangle, \mathsf{r}\rangle\rangle) \xleftarrow{cl2} fact(id_{n_2}) \ .$$

Consider the goal $fact(\langle 3, id_{n_1}\rangle) : n_1$, corresponding to the query `fact(3,A)`. A common reindexing of the arrows $M_{\{X,Y\}}(X+1,(X+1)*Y) = n_2 \xrightarrow{\langle +\langle \mathsf{I},!_{n_1}1\rangle, \times\langle +\langle \mathsf{I},!_{n_1}1\rangle,\mathsf{r}\rangle\rangle} n_2$ and $M_{\{A\}}(3,A) = \langle 3, id_{n_1}\rangle : n_1 \to n_1$ is the pair (r_1, s_1) where

$$r_1 : n_1 \to n_1 = \times\langle 3, id_{n_1}\rangle \qquad s_1 : n_1 \to n_2 = \langle 2, id_{n_1}\rangle \ ,$$

representing the substitutions $\{X/2, A/3 \times Y\}$. This corresponds to the derivation step

$$fact(\langle 3, id_{n_1}\rangle) \xrightarrow{r_1, s_1, 1, cl2} fact(\langle 2, id_{n_1}\rangle) \ .$$

Continuing this way, we have

$$fact(\langle 2, id_{n_1}\rangle) \xrightarrow{r_2, s_2, 1, cl2} fact(\langle 1, id_{n_1}\rangle) \xrightarrow{r_3, s_3, 1, cl2} fact(\langle 0, id_{n_1}\rangle) \xrightarrow{r_4, s_4, 1, cl1} \top \ ,$$

where $r_2 = \times\langle 2, id_{n_1}\rangle : n_1 \to n_1$, $r_3 = \times\langle 1, id_{n_1}\rangle = id_{n_1} : n_1 \to n_1$ and $r_4 = 1 : n_0 \to n_1$. This gives a derivation of $fact(\langle 3, id_{n_1}\rangle)$ with answer $6 : n_0 \to n_1$.

In the category of proofs $\mathcal{F}_P$, we have a corresponding proof of $fact(\langle 3,6\rangle)$ in $\mathcal{F}_P(n_0)$, given by

$$\top \xrightarrow{cl_1} fact(\langle 0,1\rangle) \xrightarrow{\langle 0,1\rangle^\sharp cl_2} fact(\langle 1,1\rangle) \xrightarrow{\langle 1,1\rangle^\sharp cl_2} fact(\langle 2,2\rangle) \xrightarrow{\langle 2,2\rangle^\sharp cl_2} fact(\langle 3,6\rangle) \ ,$$

where cl_1 and cl_2 denote the derivation steps $\xrightarrow{id_{n_0}, id_{n_0}, 1, cl1}$ and $\xrightarrow{id_{n_2}, id_{n_2}, 1, cl2}$ respectively. ■

We briefly sketch a simple semantics for the preceding example, to illustrate one of the categorical semantics treated in the next section.

Example 4.2 Taking $\mathbb{C}$ to be the Lawvere Algebraic Theory of the preceding example, we now interpret types and terms (the objects and arrows of $\mathbb{C}$) by applying the Yoneda embedding (see Equation 3.5). Goals $\mathbf{G}$ in a fiber over the type n_k are mapped to a subfunctor of $Hom(\_, n_k)$. In particular, since the predicate *fact* lives in the fiber over n_2, we have $[\![fact]\!] \subseteq Hom(\_, n_2)$. As with Herbrand models [56] there is a least Yoneda interpretation which is a model of the program, in which, for example, $[\![fact]\!](n_0) = \{\langle 0,1\rangle, \langle 1,1\rangle, \langle 0,1\rangle, \langle 2,2\rangle\langle 3,6\rangle, \langle 4,24\rangle, \ldots\}$. This interpretation is also the least fixed point of a suitably defined operator. The details are discussed in Sections 5 and 6. ■

5. The New Framework

In the previous section, we have given a detailed example of how an indexed premonoidal category can be constructed, according to specified signatures and data type information. Now we pass to the general foundation that this example suggests and give an axiomatic presentation of the semantics of a program P which lives in an ambient indexed category $\mathcal{P}$.

5.1. Syntax

In the following we introduce several kinds of indexed categories we call doctrines [46]. This is a bit of an abuse of language, since a doctrine is generally understood to be an indexed category where reindexing functors have left adjoints, and this property does not always hold for our doctrines. However, we have chosen this terminology to emphasize the relationship between indexed categories used for the syntax and the semantics, where we make use of true doctrines.

Definition 5.1 (Logic programming doctrine) *An* LP doctrine *(logic programming doctrine) is an indexed category $\mathcal{P}$ over a base category $\mathbb{C}$ each fiber $\mathcal{P}\sigma$ of which has a strict premonoidal structure $(\otimes_\sigma, \top_\sigma)$ which is preserved on the nose by reindexing functors.*

For each $\sigma \in |\mathbb{C}|$, 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 $t : \rho \to \sigma$ in $\mathbb{C}$, $t^\sharp\mathbf{G}$ is an *instance* of $\mathbf{G}$. The premonoidal tensor $\otimes_\sigma$ builds conjunctions of goals of sort σ, while $\top_\sigma$ corresponds to the empty goal.

We write $\mathbf{G} : \sigma$ and $f : \sigma$ as a short form for $\mathbf{G} \in |\mathcal{P}\sigma|$ and $f \in \mathcal{P}\sigma$. Given an LP doctrine $\mathcal{P}$, a *clause* (of sort σ) is a tuple $(cl, \sigma, \mathbf{Hd}, \mathbf{Tl})$ where $\sigma \in |\mathbb{C}|$, $\mathbf{Tl} : \sigma$, $\mathbf{Hd} : \sigma$ and cl is a label which uniquely identifies the clause[2]. In the following we will write this clause as $\mathbf{Hd} \xleftarrow{cl:\sigma} \mathbf{Tl}$, and we will omit σ when it is clear from the context. A set of clauses is called *program*.

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 a set of deductions that can be performed. These local deductions do not depend on the program we want to execute. This corresponds to so-called *built-in* predicates with built-in deduction steps. Program clauses yield new deductions in addition to the proofs in the fibers.

What we mean by state here is quite broad: it can be the value of some global storage, a local constraint, or just the current tuple of free variables (as in the standard hyperdoctrinal semantics for logic).

Example 5.2 (Pure logic programs) Assume given a first order signature (Σ, Π). We build the category $\mathbb{S}_\Sigma$, which is the free Lawvere Algebraic Theory [52] generated by Σ. We also assume fixed a denumerable sequence $v_1, \ldots, v_n, \ldots$ of variables.

Objects of $\mathbb{S}_\Sigma$ are natural numbers (with zero), while arrows from n to m are substitutions $\{v_1/t_1, \ldots, v_m/t_m\}$ where $t_1, \ldots, t_m$ are terms built from the set of variables $\{v_1, \ldots, v_n\}$. Arrows compose according to the standard notion of composition of substitutions: if $\theta : n \to n'$ and $\theta' : n' \to n''$, then $\theta\theta' : n \to n''$ is given by $(\theta \diamond \theta')(v_i) = v_i\theta'\theta$ for each $i \in \{1, \ldots, n''\}$.

Then, we build an LP doctrine $\mathcal{P}_\Sigma : \mathbb{S}_\Sigma{}^{\mathrm{o}} \to \mathsf{Cat}$ such that

[2]Labels make it possible to have two different clauses which have the same body and tail.

- for each $n \in \mathbb{N}$, $\mathcal{P}_\Sigma(n)$ is the discrete category of syntactic goals (i.e. possibly empty sequences of atoms) built from the variables $v_1, \ldots, v_n$ and symbols in Σ and Π;
- for each $\theta : n \to m$, $\mathcal{P}_\Sigma(\theta)$ is the functor mapping a goal $\mathbf{G}$ to $\mathbf{G}\theta$;
- the premonoidal structure in $\mathcal{P}_\Sigma(n)$ is given by $(\otimes_n, \top_n)$ where $\otimes_n$ is concatenation of sequences and $\top_n$ is the empty sequence. The structure is in fact monoidal.

It is evident that $\mathcal{P}_\Sigma$ is a functor and that $\mathcal{P}_\Sigma(\theta)$ preserves the monoidal structure.

Alternatively, we may define a display category $\mathbb{J}$ with the elements of Π as discrete objects, and $\delta : \mathbb{J} \to \mathbb{S}_\Sigma$ which maps every predicate symbol p to its arity. Then $\mathcal{P}_\Sigma$ would be isomorphic to $\mathcal{P}_\mathbb{J}$.

A clause in $\mathcal{P}_\Sigma$ is a pair of goals. This is a straightforward generalization of the standard notion of clause in logic programming, since we also admit clauses whose head is not atomic. It is also quite easy to extend these definitions to work with a many-sorted signature Σ. ■

When we define the concept of model for a program, below, it will be clear that thus far we have not imposed any conditions on the possible meaning of predicates. However, we can choose categories with more structure for fibers in general, allowing us to constrain permissible interpretations. A simple instance is considered in the following example.

Example 5.3 (Symmetric predicates) Given a discrete display structure $\mathbb{J}$ over the FP category $\mathbb{C}$, assume we have a $p \in |\mathbb{J}|$ of sort $\rho \times \rho$, and we want to encode in the syntactic doctrine the property that p is symmetric. Then, we freely adjoin to $\mathcal{P}_\mathbb{J}$ the following arrow in the fiber $\rho \times \rho$:

$$\mathit{refp} : p(id_{\rho\times\rho}) \to p(\langle \pi_2, \pi_1 \rangle) \ .$$

We call $\mathcal{P}_\mathbb{J}^{\mathit{refp}}$ the new LP doctrine we obtain. The intuitive meaning of the adjoined arrow is evident. We will see in the following sections how it formally affects the semantics of a program in $\mathcal{P}_\mathbb{J}^{\mathit{refp}}$. A more formal definition of $\mathcal{P}_\mathbb{J}^{\mathit{refp}}$ is given in Section 8.1. ■

A particular case of LP doctrine, and arguably the most important one, is when goals are essentially sequences of atomic goals, and the only meaningful arrows are those whose targets are atomic goals.

Definition 5.4 (Atomic LP doctrines) *An LP doctrine $\mathcal{P} : \mathbb{C}^{\mathrm{o}} \to \mathsf{Cat}$ is* atomic *when, for each σ in the base category, there is a set $A_\sigma \subseteq |\mathcal{P}\sigma|$ of objects, in the following called* atomic goals*, such that:*

1. *reindexing functors preserve atomic goals (i.e. if $r : \sigma \to \rho$ and $A \in A_\rho$, then $r^\sharp A \in A_\sigma$);*

2. *for every goal* $\mathbf{G} \in \mathcal{P}\sigma$ *there is an unique (possibly empty) sequence* $A_1, \ldots, A_n$ *of atomic goals of sort* σ *such that* $\mathbf{G} = A_1 \otimes_\sigma \cdots \otimes_\sigma A_n$*, with the proviso that if* $n = 0$*, then* $\mathbf{G} = \top_\sigma$*;*
3. *for every arrow* $f : \mathbf{G}' \to \mathbf{G}_1 \otimes \mathbf{G}_2$ *there are arrows* $f_1 : \mathbf{G}'_1 \to \mathbf{G}_1$ *and* $f_2 : \mathbf{G}'_2 \to \mathbf{G}_2$ *such that either* $f = (f_1 \otimes \mathbf{G}'_2) \diamond (\mathbf{G}_1 \otimes f_2)$ *or* $f = (\mathbf{G}'_1 \otimes f_2) \diamond (f_1 \otimes \mathbf{G}_2)$*;*
4. *for every arrow* $f : \mathbf{G}' \to \top$*,* f *is the identity* $id_\top$*.*

A program P *over* $\mathcal{P}$ *is atomic when* $\mathcal{P}$ *is an atomic LP doctrine and all the heads of clauses in* P *are atomic goals.*

Note that in an atomic LP doctrine, $\top$ is never considered to be an atomic goal, since this would violate the second condition in Definition 5.4. In fact, we could write $\top$ as $\top$, $\top \otimes \top$, $\top \otimes \top \otimes \top$ and so on.

Example 5.5 $\mathcal{P}_\Sigma$ and $\mathcal{P}_\mathbb{J}$ are atomic LP doctrines, with obvious definitions for the set of atomic goals. In $\mathcal{P}_\Sigma$ it is enough to let A_σ be the set of atomic goals, according to the standard definition. Moreover, all standard logic programs are atomic programs. In $\mathcal{P}_\mathbb{J}$, we let A_σ be the set of all goals of the form $p(t)$ where $p \in |\mathbb{J}|$ and $\mathsf{cod}(t) = \delta(p)$. ■

Example 5.6 (Generic predicates) Another atomic LP doctrine is introduced in [23, 24]. Given a FP category $\mathbb{C}$ and $\vec{\sigma} = \sigma_1, \ldots, \sigma_n$ a sequence of objects of $\mathbb{C}$, the indexed monoidal category $\Pi_{\vec{\sigma}} : \mathbb{C}^{\mathrm{o}} \to \mathsf{Cat}$ of *generic predicates* of sort $\vec{\sigma}$ is defined as follows. Each fiber $\Pi_{\vec{\sigma}}(\rho)$ has objects the members of $\wp_f(Hom(\rho, \sigma_1)) \times \cdots \times \wp_f(Hom(\rho, \sigma_n))$, i.e. sequences $S = S_1, \ldots, S_n$ where each S_i is a finite set of arrows from ρ to σ_i, further endowed with the poset operation of pointwise containment: $S \leq T$ iff $S_i \subseteq T_i$ for each $i \in \{1, \ldots, n\}$.

The monoidal operator $S_1 \otimes_\rho S_2$ is the pointwise union of the two sequences. Note that $\otimes_\rho$ is actually a strict product in the fiber $\Pi_{\vec{\sigma}}(\rho)$, hence the ordering of atoms in a goal is lost. The same happens to the multiplicity of atoms.

The action of $\Pi_{\vec{\sigma}}$ on the arrow $r : \rho_1 \to \rho_2$ is given by $\Pi_b(r)(S) = rS$ where rS is obtained by replacing each $t : \rho_2 \to \sigma_i$ in S with rt.

The idea underlying the definition of $\Pi_{\vec{\sigma}}$ is that if $\vec{\sigma}$ has length n, there are n generic predicates of sorts $\sigma_1, \ldots, \sigma_n$. Adopting the notation we used for the LP doctrine $\mathcal{P}_\mathbb{J}$, we would call $p_1, \ldots, p_n$ such generic predicates. Each sequence $S = S_1, \ldots, S_n$ represents a goal $\mathbf{G}$ such that, for each $t : \rho \to \sigma_i \in S_i$, $\mathbf{G}$ has a corresponding conjunct $p_i(t)$. This idea may be made precise by defining an indexed monoidal functor $\langle id_\mathbb{C}, \tau \rangle$ from $\mathcal{P}_\mathbb{J}$ to $\Pi_{\vec{\sigma}}$, where $\mathbb{J}$ is the discrete category whose objects are the natural numbers $1, \ldots, n$ and $\delta : \mathbb{J} \to \mathbb{C}$ maps each i to σ_i. ■

5.2. Operational Semantics

The operational semantics of logic programs is traditionally based on goal rewriting induced by program clauses. This is also how ours is defined, but we need to make allowances for the fact that, in our case, goals are not syntactic entities, but objects that live in an LP doctrine $\mathcal{P}$. Moreover, we also want to

use arrows in the fibers of $\mathcal{P}$ to rewrite goals, not just clauses: rewriting with respect to an arrow corresponds to applying a built-in definition of a predicate and a built-in deduction rule.

First consider the case where the goal is $\mathbf{G} : \sigma$ and $f : \mathbf{G} \leftarrow \mathbf{G}' \in \mathcal{P}\sigma$ an arrow in $\mathcal{P}$. Then we just treat f as a clause in a standard logic program, and rewrite $\mathbf{G}$ to $\mathbf{G}'$. Now, suppose $f : r^\sharp\mathbf{G} \leftarrow \mathbf{G}' \in \mathcal{P}\rho$. Although the head of the clause is not exactly the same as the current goal, we still want to rewrite $\mathbf{G}$ to $\mathbf{G}'$. This is because goals in logic programs are implicitly existential, hence when we look for a proof for $\mathbf{G}$, we are actually looking for a proof of any instance of $\mathbf{G}$. Therefore, the general form of rewriting rule could be stated as follows: *if* $\mathbf{G} : \sigma$ *and* $f : r^\sharp\mathbf{G} \leftarrow \mathbf{G}' \in \mathcal{P}\rho$ *for some* $r : \rho \to \sigma$*, then rewrite* $\mathbf{G}$ *to* $\mathbf{G}'$.

This is enough when we want to rewrite a goal w.r.t. an arrow. Note that, if we have a goal $s^\sharp\mathbf{G} \in \mathcal{P}\sigma$ and an arrow $f : r^\sharp\mathbf{G} \leftarrow \mathbf{G}' \in \mathcal{P}\rho$ it may be the case that $r^\sharp\mathbf{G}$ is not an instance of $s^\sharp\mathbf{G}$. Then we need to find a pair of arrows $\langle r' : \alpha \to \rho, s' : \alpha \to \sigma\rangle$ such that $r'^\sharp r^\sharp\mathbf{G} = s'^\sharp s^\sharp\mathbf{G}$. The pair $\langle r', s'\rangle$ is the counterpart in our framework of a unifier of $r^\sharp\mathbf{G}$ and $s^\sharp\mathbf{G}$. Since $\mathcal{P}$ is an indexed category, there is an arrow $r'^\sharp f : r'^\sharp r^\sharp\mathbf{G} \leftarrow r'^\sharp\mathbf{G}'$ where $r'^\sharp r^\sharp\mathbf{G}$ is an instance of $s^\sharp\mathbf{G}$, hence we may rewrite $s^\sharp\mathbf{G}$ to $r'^\sharp\mathbf{G}'$ according to the rewriting rule above. Moreover, given the goal $\mathbf{G}_1 \otimes \mathbf{G} \otimes \mathbf{G}_2 : \sigma$ and the arrow $f : r^\sharp\mathbf{G} \leftarrow \mathbf{G}'$, thanks to the premonoidal structure on the fiber, we also have an arrow $r^\sharp\mathbf{G}_1 \otimes f \otimes r^\sharp\mathbf{G}_2 : r^\sharp(\mathbf{G}_1 \otimes \mathbf{G} \otimes \mathbf{G}_2) \leftarrow r^\sharp\mathbf{G}_1 \otimes \mathbf{G}' \otimes r^\sharp\mathbf{G}_2$, hence we may rewrite the original goal to $r^\sharp\mathbf{G}_1 \otimes \mathbf{G}' \otimes r^\sharp\mathbf{G}_2$. In the case of conjunctive goals, the rewriting rule behaves as expected from standard logic programming.

When we consider rewriting w.r.t. a clause, we are tempted to say: *if* $\mathbf{G} : \sigma$ *and* $r^\sharp\mathbf{G} \xleftarrow{cl:\rho} \mathbf{Tl} \in P$ *for some* $r : \rho \to \sigma$*, then rewrite* $\mathbf{G}$ *to* $\mathbf{Tl}$. However, this does not work very well. For example, we would like to rewrite the goal $r'^\sharp r^\sharp\mathbf{G} \otimes \mathbf{G}'$ w.r.t. the clause cl above in order to obtain $r'^\sharp\mathbf{Tl} \otimes \mathbf{G}'$ (just as $p_1(a), p_2(b)$ is rewritten via the clause $p_1(x) \leftarrow q(x)$ into $q(a), p_2(b)$ in pure logic programs). With the definition above we would need a clause $r'^\sharp r^\sharp\mathbf{G} \otimes \mathbf{G}' \longleftarrow r'^\sharp\mathbf{Tl} \otimes \mathbf{G}'$, which is not guaranteed to exist. Therefore, we need to consider not only the clauses in the program, but also all the other clauses which may be obtained from the former by applying reindexing and premonoidal tensor.

Definition 5.7 (Formal clauses) *A* formal clause *of sort* σ *is a tuple* $fcl = \langle \mathbf{G}_a, t, cl, \mathbf{G}_b\rangle$ *where* $\mathbf{G}_a : \sigma,\ \mathbf{G}_b : \sigma,\ t : \sigma \to \rho$ *and* $cl : \mathbf{Hd} \leftarrow \mathbf{Tl}$ *is a clause of sort* ρ*. We write* $fcl : \mathbf{G}' \leftarrow \mathbf{G}$ *iff* $\mathbf{G} = \mathbf{G}_a \otimes t^\sharp\mathbf{Tl} \otimes \mathbf{G}_b$ *and* $\mathbf{G}' = \mathbf{G}_a \otimes t^\sharp\mathbf{Hd} \otimes \mathbf{G}_b$.

Given $s : \rho \to \sigma$, we define a reindexing operator $s^\sharp$ which maps a formal clause $\langle \mathbf{G}_a, t, cl, \mathbf{G}_b\rangle : \mathbf{G}' \leftarrow \mathbf{G}$ of sort σ to $\langle s^\sharp\mathbf{G}_a, s \diamond t, cl, s^\sharp\mathbf{G}_b\rangle : s^\sharp\mathbf{G}' \leftarrow s^\sharp\mathbf{G}$. Moreover, for each goal $\mathbf{G}_1 : \sigma$ we define a tensor product $\mathbf{G}_1 \otimes_\sigma \_$ which maps $\langle \mathbf{G}_a, t, cl, \mathbf{G}_b\rangle : \mathbf{G}' \leftarrow \mathbf{G}$ of sort σ to $\langle \mathbf{G}_1 \otimes_\sigma \mathbf{G}_a, t, cl, \mathbf{G}_b\rangle : \mathbf{G}_1 \otimes_\sigma \mathbf{G}' \leftarrow \mathbf{G}_1 \otimes_\sigma \mathbf{G}$. An analogous tensor product may be defined for conjunction on the right, i.e. $\_ \otimes_\sigma \mathbf{G}_1$. It is easy to check that formal clauses with reindexing and tensor products form an indexed premonoidal graph.

When we use a clause $cl : \sigma$ in a context where a formal clause would be expected, cl should be understood as $\langle \top_\sigma, id_\sigma, cl, \top_\sigma \rangle$.

The informal rewrite rule "*if* $\mathbf{G} : \sigma$ *and* $cl : r^\sharp \mathbf{G} \leftarrow \mathbf{Tl} \in P$ *for some* $r : \rho \to \sigma$*, then rewrite* $\mathbf{G}$ *to* $\mathbf{Tl}$" may be changed to "*if* $\mathbf{G} : \sigma$ *and* $fcl : r^\sharp \mathbf{G} \leftarrow \mathbf{Tl}$ *is a formal clause for some* $r : \rho \to \sigma$*, then rewrite* $\mathbf{G}$ *to* $\mathbf{Tl}$", and everything works in the same way as for rewriting w.r.t. an arrow.

Definition 5.8 (Categorical derivations) *Given a program* P *over* $\mathcal{P}$*, we define a labeled transition system* $(\biguplus_{\sigma \in |\mathbb{C}|} |\mathcal{P}\sigma|, \rightsquigarrow)$ *with goals as objects, according to the following* backchain *rule:*

$$\mathbf{G} : \sigma \xrightarrow{\langle r, f \rangle} \mathbf{Tl} : \rho \iff r : \rho \to \sigma \in \mathbb{C}, \text{ and } f : r^\sharp \mathbf{G} \leftarrow \mathbf{Tl} \ , \tag{5.1}$$

where f *is either an arrow in* $\mathcal{P}\rho$ *or a formal clause of sort* ρ*. A* categorical derivation *is a (possibly empty) derivation in this transition system.*

The pairs $\langle r, f \rangle$ which label the transitions are called *reduction pairs* and they uniquely identify a single step. If we want to distinguish between steps $\langle r, f \rangle$ where f is an arrow or a formal clause, we speak of arrow reduction pairs and clause reduction pairs respectively.

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

$$\mathbf{G}_0 \overset{l_0}{\rightsquigarrow} \mathbf{G}_1 \overset{l_1}{\rightsquigarrow} \cdots \xrightarrow{l_{i-2}} \mathbf{G}_{i-1} \xrightarrow{l_{i-1}} \mathbf{G}_i \ , \tag{5.2}$$

we write $\mathbf{G}_0 \overset{d}{\rightsquigarrow}{}^* \mathbf{G}_i$ where $d = l_0 \cdots l_{i-1}$ is the string obtained concatenating all the labels. Note that $d \neq \epsilon$ (the empty sequence) uniquely identifies the corresponding sequence of goals. We will write $\epsilon_{\mathbf{G}}$ for the empty derivation starting from the goal $\mathbf{G}$. A derivation $d : \mathbf{G} \rightsquigarrow^* \top$ is a *successful derivation.*

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

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

The *correct answers* for a goal $\mathbf{G}$ are all the arrows $\mathsf{answer}(d)$ for some successful derivation d of $\mathbf{G}$.

We want to point out that categorical derivations correspond to standard SLD derivations where a generic unifier, instead of the most general one, is chosen at each step. Therefore, our definition of correct answers corresponds to correct answers for standard logic programming, not to computed answers. More on these correspondences may be found in Example 5.17.

5.2.1. Most General Derivations

Some derivations are more general than others. Consider in $\mathcal{P}_\Sigma$ the goal $p(v_1, v_2)$ and assume there is an arrow $f : p(2, v_2) \leftarrow q(v_2) \in \mathcal{P}_\Sigma(2)$. Obviously we also have an arrow $f' : p(2, 3) \leftarrow q(3)$ where $f' = \theta^\sharp f$ and $\theta = \{v_2/3\}$. Therefore both $d_1 = p(v_1, v_2) \xrightarrow{\langle \{v_1/2\}, f \rangle} q(v_2)$ and $d_2 = p(v_1, v_2) \xrightarrow{\langle \{v_1/2, v_2/3\}, f' \rangle} q(3)$ are

valid transitions. However, the first one is more general, since d_2 factors through d_1 as $p(v_1, v_2) \xrightarrow{\langle \{v_1/2\}, f \rangle} q(v_2) \xrightarrow{\langle \{v_2/3\}, id \rangle} q(3)$, but d_1 does not factor through d_2. Note that if we consider the reduction pairs used in d_1 and d_2, the fact that d_1 is more general that d_2 is reflected by the existence of the substitution $\theta = \{v_2/3\}$ such that $\langle \{v_1/2, v_2/3\}, f' \rangle = \langle \{v_1/2\}\theta, \theta^\sharp f' \rangle$.

Definition 5.9 (Category of reduction pairs) *Reduction pairs for a goal* $\mathbf{G}$ *form a category* $\mathsf{Red}_{\mathbf{G}}$*. An arrow from* $\langle r_1, f_1 \rangle$ *to* $\langle r_2, f_2 \rangle$ *is an arrow* $t \in \mathbb{C}$ *such that* $r_1 = t \diamond r_2$ *and* $f_1 = t^\sharp f_2$*. Arrows compose as they do in* $\mathbb{C}$*.*

Among reduction pairs, we are interested to those which are *maximal*, according to the following definition. A *maximal object* in category $\mathbb{C}$ is an object A such that, for each $f : A \to B$, there exists an unique $g : B \to A$ such that $fg = id_A$.

Proposition 5.10 *If* $\mathbb{C}$ *has terminal objects, they are the only maximal objects.*

Proof. First we prove that $\mathbf{1}$ is maximal. If $f : \mathbf{1} \to B$, there exists an unique $!_B : B \to \mathbf{1}$ and $f!_B = id_{\mathbf{1}}$. Now assume A is maximal, i.e. there exists an arrow $f : \mathbf{1} \to A$ such that $!_A f = id_A$. But $f!_A = id_{\mathbf{1}}$, hence f is an iso. ■

A *most general* reduction pair for $\mathbf{G}$ is a maximal object in $\mathsf{Red}_{\mathbf{G}}$. Note that the use of the term "most general", here and in the rest of the paper, is improper. In the general case, $\mathsf{Red}_{\mathbf{G}}$ has no terminators, and a most general reduction pair is not unique, not even up to iso.

Example 5.11 (Arrow reduction pairs) In the categories $\mathcal{P}_\Sigma$ and $\mathcal{P}_\mathbb{J}$ the only arrows in the fibers are identities. This means that the only arrow reduction pairs for $\mathbf{G} : \sigma$ are the tuples $\langle r, id_{r^\sharp \mathbf{G}} \rangle$ for any r with $\mathsf{cod}(r) = \sigma$. Moreover, the reduction pair $\langle id_\sigma, id_{\mathbf{G}} \rangle$ is maximal in $\mathsf{Red}_{\mathbf{G}}$, hence it is a most general reduction pair. If we consider the category $\mathcal{P}_\mathbb{J}^{refp}$, other reduction pairs are generated by the arrow *refp*, such as $\langle r, r^\sharp(refp) \rangle$ for any r such that $\mathsf{cod}(r) = \sigma \times \sigma$. ■

Remark 5.12 (Maximal objects and IPOs) Maximality (better, the dual notion of maximality) is related to the concept of *idem pushout* introduced in [54]. Idem pushouts are used with reaction systems in order to give a precise categorical definition of the notion of "just large enough". There, the aim was to characterize, given an agent a, the minimal contexts $F[\_]$ such that $F[a]$ may react. Here, we look for the "least instantiated" reduction pair for a goal $\mathbf{G}$.

An object $A \in |\mathbb{C}|$ is *minimal* when, for each arrow $f : B \to A$, there exists an unique arrow $g : B \to A$ such that $gf = id_B$. Given a span $S = \langle t_1 : A \to B_1, t_2 : A \to B_2 \rangle$, let $CSp(S)$ be the category of cospans $\langle r_1 : B_1 \to C, r_2 : B_2 \to C \rangle$ such that $t_1 r_1 = t_2 r_2$. It is possible to prove that the commuting

diagram

$$\begin{array}{ccc} A & \xrightarrow{t_1} & B_1 \\ {\scriptstyle t_2}\downarrow & & \downarrow{\scriptstyle r_1} \\ B_2 & \xrightarrow[r_2]{} & C \end{array}$$

is an *idem pushout* (IPO) iff $\langle r_1, r_2 \rangle$ is minimal in $CSp(\langle t_1, t_2 \rangle)$. ■

If we restrict the backchain rule to most general reduction pairs, we get a new transition system $(\biguplus_{\sigma \in |\mathbb{C}|} |\mathcal{P}\sigma|, \rightsquigarrow_g)$ and a corresponding notion of *most general* (m.g.) categorical derivation. In the following, when not otherwise stated, everything we say about categorical derivations can be applied to m.g. ones. In particular, if d is a most general successful derivation of the goal $\mathbf{G}$, then $\mathsf{answer}(d)$ is called a *computed answer* of $\mathbf{G}$.

5.2.2. Formal Clauses and Unifiers

Here we want to show that, for pure logic programs, categorical SLD derivation is faithful w.r.t. standard SLD derivation. We begin to show the relationship which holds between the concepts of unifier, which is used in the standard definition of SLD derivation, and reduction pair, which is used in our framework.

First of all, we introduce a categorical generalization of the notion of unifier, along the lines of [76].

Definition 5.13 (Unifier) *A unifier for goals $\mathbf{G}_1 : \sigma_1$ and $\mathbf{G}_2 : \sigma_2$ in an LP doctrine $\mathcal{P}$ is a span $\langle t_1, t_2 \rangle$ of arrows on the base category such that $t_1 : \alpha \to \sigma_1$, $t_2 : \alpha \to \sigma_2$ and $t_1{}^\sharp \mathbf{G}_1 = t_2{}^\sharp \mathbf{G}_2$.*

Like reduction pairs, unifiers for a pair of goals form a category $\mathsf{Unif}_{\mathbf{G}_1, \mathbf{G}_2}$ 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 $f \diamond r_1 = t_1$ and $f \diamond r_2 = t_2$. An mgu (*most general unifier*) for goals $\mathbf{G}_1 : \sigma_1$ and $\mathbf{G}_2 : \sigma_2$ in an LP doctrine $\mathcal{P}$ is a maximal object in $\mathsf{Unif}_{\mathbf{G}_1, \mathbf{G}_2}$.

Example 5.14 (Standard mgu) Consider the indexed categories $\mathcal{P}_\Sigma$ and $\mathcal{P}_\mathbb{J}$. Given a predicate symbol p of sort σ and atomic goals $p(t_1) : \sigma_1$ and $p(t_2) : \sigma_2$, a unifier is a pair of arrows $r_1 : \alpha \to \sigma_1$ and $r_2 : \alpha \to \sigma_2$ such that the following diagram commutes:

$$\begin{array}{ccc} \alpha & \xrightarrow{r_1} & \sigma_1 \\ {\scriptstyle r_2}\downarrow & & \downarrow{\scriptstyle t_1} \\ \sigma_2 & \xrightarrow[t_2]{} & \sigma \end{array} \tag{5.3}$$

This is exactly the definition of unifier for renamed apart terms t_1 and t_2 given in [4], which corresponds to unifiers in the standard syntactical sense. Moreover, the span $\langle r_1, r_2 \rangle$ is terminal, hence maximal, when (5.3) is a pullback diagram. ■

Theorem 5.15 *If $\mathbf{G} = A_1, \ldots, A_n$ and P is atomic, a clause reduction pair of $\mathbf{G}$ has the form $\langle r, fcl\rangle$ where $fcl = \langle r^\sharp A_1 \otimes \cdots \otimes r^\sharp A_{i-1}, s, cl, r^\sharp A_{i+1} \otimes \cdots \otimes r^\sharp A_n\rangle$ and $\langle r, s\rangle$ is an unifier of A_i and $\mathbf{Hd}$, the head of cl. It is maximal iff $\langle r, s\rangle$ is maximal.*

Proof. The proof of this theorem may be found in the Appendix. ∎

Note that, in the hypothesis of Theorem 5.15, if $\langle r, s\rangle$ is a terminal element in $\mathsf{Unif}_{A_i,\mathbf{Hd}}$ the corresponding reduction pair does not need to be terminal in $\mathsf{Red}_{\mathbf{G}}$. For example, if there exists an index $j \neq i$ and an unifier $\langle r', s'\rangle$ of A_j and the head of the clause cl, we get another reduction pair $\langle r', fcl'\rangle$ with

$$fcl' = \langle r'^\sharp A_1 \otimes \cdots \otimes r'^\sharp A_{j-1}, s', cl, r'^\sharp A_{j+1} \otimes \cdots \otimes r'^\sharp A_n\rangle \ . \tag{5.4}$$

It is obvious that there are no arrows from $\langle r', fcl'\rangle$ to $\langle r, fcl\rangle$. Only maximality is preserved.

Example 5.16 (SLD derivations for atomic programs) If P is an atomic program over the LP doctrine $\mathcal{P}$, it is possible to adopt a simpler notation for SLD derivations, which is the one used in Section 4.1. An SLD step $\mathbf{G} \xrightarrow{\langle r, fcl\rangle} \mathbf{G}'$ where $r : \rho \to \sigma_1$ and $\mathbf{G} = A_1 \otimes \cdots \otimes A_n$ of sort σ_1, is uniquely determined by a choice of an atom A_i to reduce, a clause cl (or arrow f in the fibers) of sort σ_2, and an arrow $s : \rho \to \sigma_2$ such that $\langle r, s\rangle$ is an unifier of A_i and the head of cl (or the codomain of f). Therefore, we may write $\mathbf{G} \xrightarrow{\langle r, s, i, cl\rangle} \mathbf{G}'$ to actually mean $\mathbf{G} \xrightarrow{\langle r, fcl\rangle} \mathbf{G}'$ and $fcl = \langle r^\sharp A_1 \otimes \cdots \otimes r^\sharp A_{i-1}, s, cl, r^\sharp A_{i+1} \otimes \cdots \otimes r^\sharp A_n\rangle$. ∎

Example 5.17 (Standard SLD derivations) Consider an atomic program P in the syntactic doctrine $\mathcal{P}_\Sigma$, a goal $p_1(\vec{t_1}), \ldots, p_n(\vec{t_m})$ and a clause $p_i(\vec{t}) \xleftarrow{cl} \mathbf{G}$. The most general unifier $\langle r, s\rangle$ of $p_i(\vec{t})$ and $p_i(\vec{t_i})$ corresponds to a most general (i.e. maximal) reduction pair which yields the following derivation step:

$$\begin{aligned} p_1(\vec{t_1}), \ldots, p_{i-1}(\vec{t}_{i-1}), p_i(\vec{t_i}), p_{i+1}(\vec{t}_{i+1}), \ldots, p_n(\vec{t_n}) &\xrightarrow{\langle r, fcl\rangle}_g \\ r^\sharp p_1(\vec{t_1}), \ldots, r^\sharp p_{i-1}(\vec{t}_{i-1}), s^\sharp \mathbf{G}, r^\sharp p_{i+1}(\vec{t}_{i+1}), \ldots, r^\sharp p_m(\vec{t_m}) \ . \end{aligned} \tag{5.5}$$

In the doctrine $\mathcal{P}_\Sigma$, this strictly corresponds to a step of the standard SLD derivation procedure. However, in our categorical framework, it is possible to reduce w.r.t. one of the identity arrows in the fibers. Therefore, if $\mathbf{G}$ is a goal of arity n,

$$\mathbf{G} \xrightarrow{\langle id_n, id_{\mathbf{G}}\rangle}_g \mathbf{G} \tag{5.6}$$

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 $\mathbf{G}$, the set

$$\mathsf{answer}\{d \mid d : \mathbf{G} \rightsquigarrow_g{}^* \top\} \tag{5.7}$$

is exactly the set of computed answers for the goal $\mathbf{G}$ returned by standard SLD resolution.

If we consider a derivation in $\rightsquigarrow$ instead of $\rightsquigarrow_g$, at every step we may perform an arbitrary instantiation of the goal we are resolving. It is like having a variant of SLD resolution which selects generic unifiers instead of mgus. Then

$$\mathsf{answer}\{d \mid d : \mathbf{G} \rightsquigarrow^* \top\} \tag{5.8}$$

is the set of correct answers for the goal $\mathbf{G}$.

If the program P is not atomic, things are different. A clause $p_1(\vec{t_1}), p_2(\vec{t_2}) \overset{cl}{\leftarrow} \mathbf{G}$ does not have a counterpart in standard logic programs. It is *not* equivalent to the pair of clauses $p_1(\vec{t_1}) \overset{cl}{\leftarrow} \mathbf{G}$ and $p_2(\vec{t_2}) \overset{cl}{\leftarrow} \mathbf{G}$. If we had chosen finite products as the structure for managing conjunction, then we would have projection arrows $\pi_i : p_1(\vec{t_1}), p_2(\vec{t_2}) \to p_i(\vec{t_i})$ and we could resolve a goal $p_1(\vec{t_1})$ with π_1 and later with cl to obtain $\mathbf{G}$. With a premonoidal structure we do not get projection arrows. Therefore, we cannot rewrite $p_1(\vec{t_1})$ using the clause $p_1(\vec{t_1}), p_2(\vec{t_2}) \overset{cl}{\leftarrow} \mathbf{G}$. ■

5.3. Declarative Semantics

One of the main aims of this treatment is to consider extensions to definite logic programs without losing the declarative point of view. To this end, we define a straightforward declarative notion of model for a program, and show that there is a strict correspondence between models and categorical derivations.

Definition 5.18 (Interpretations) *An* interpretation *of the LP doctrine $\mathcal{P}$ in the LP doctrine $\mathcal{Q}$ is a premonoidal indexed functor $\langle F, \tau \rangle$ from $\mathcal{P}$ to $\mathcal{Q}$.*

If $[\![\_]\!] = \langle F, \tau \rangle$ is an interpretation from $\mathcal{P}$ to $\mathcal{Q}$, we write $F(e)$ as $[\![e]\!]$ for every object e or arrow e in $\mathbb{C}$. Moreover, if x is either a goal or an arrow in the fiber $\mathcal{P}\sigma$, we write $\tau_\sigma(x)$ as $[\![x]\!]_\sigma$. We also use $[\![x]\!]$ when the fiber is clear from the context. Finally, we will denote with $t^\sharp$ and $t^\flat$ the reindexing functors $\mathcal{P}(t)$ and $\mathcal{Q}(Ft)$ respectively.

Definition 5.19 (Models) *Given a program P over the LP doctrine $\mathcal{P}$, a* model *of P is a pair $([\![\_]\!], \iota)$ where $[\![\_]\!] : \mathcal{P} \to \mathcal{Q}$ is an interpretation and ι is a map from clauses (of sort σ) to arrows in $\mathcal{Q}$ (over the fiber $[\![\sigma]\!]$).*

In the following, a model $M = ([\![\_]\!], \iota)$ will be used as an alias for its constituent parts. Hence, $M(cl)$ will be the same as $\iota(cl)$ and $M_\sigma(\mathbf{G})$ the same as $[\![\mathbf{G}]\!]_\sigma$. Moreover, we define the composition of M with an interpretation N as the model $([\![\_]\!] \diamond N, \iota \diamond N)$, where $\iota \diamond N$ is the function which maps the clause cl of sort σ to $N_{[\![\sigma]\!]}(\iota(cl))$.

Models $M : \mathcal{P} \to \mathcal{Q}$ for a program P give an *and*-compositional semantics of goals *provided P is atomic*: $M(\mathbf{G}_1 \otimes_\sigma \mathbf{G}_2) = M(\mathbf{G}_1) \otimes_{M(\sigma)} M(\mathbf{G}_2)$. However, compositionality fails in general if non-atomic heads are allowed in clauses. A good example is the correct answer semantics, which maps a goal $\mathbf{G}$ to the set $\{\mathsf{answer}(d) \mid d : \mathbf{G} \rightsquigarrow^* \top\}$, in the case of the LP doctrine $\mathcal{P}_{\mathbb{J}}$ and the program $P = \{p(t_1), p(t_2) \leftarrow \top\}$. The goal $p(t_1), p(t_2)$ has a successful derivation, while

neither $p(t_1)$ nor $p(t_2)$ do. On the other hand, given the program $P = \emptyset$, the goal $p(t_1), p(t_2)$ has no successful derivations. This means that we cannot obtain the correct answers for $p(t_1), p(t_2)$ from the correct answers for $p(t_1)$ and $p(t_2)$.

Example 5.20 (Correct ground answers) Given a category $\mathbb{C}$ with terminators, define the semantic LP doctrine $\mathcal{G}_{\mathbb{C}}$ over $\mathbb{C}$ as follows.

- for each $\sigma \in |\mathbb{C}|$, $\mathcal{G}_{\mathbb{C}}(\sigma) = \wp(Hom(\mathbf{1}, \sigma))$, which is an ordered set viewed as a category;
- for each $t \in Hom_{\mathbb{C}}(\sigma, \rho)$, $\mathcal{G}_{\mathbb{C}}(t)(X) = \{t' \in Hom(\mathbf{1}, \sigma) \mid t't \in X\}$;
- the premonoidal structure on $\mathcal{G}_{\mathbb{C}}(\sigma)$ is given by set intersection as tensor product and $Hom(\mathbf{1}, \sigma)$ is the premonoidal unit.

If $\mathcal{P}$ is an LP doctrine over $\mathbb{C}$, an interpretation $[\![\_]\!] : \mathcal{P} \to \mathcal{G}_{\mathbb{C}}$ (with an identity change of base functor) maps a 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 substitutions.

A curious trivial model for every program is given by the interpretation which maps every goal $\mathbf{G} : \sigma$ to $[\![\mathbf{G}]\!]_\sigma = Hom(\mathbf{1}, \sigma)$. Since the fibers are posets, this assignment induces an unique model, mapping every clause $cl : \mathbf{Hd} \leftarrow \mathbf{Tl}$ of sort σ and every arrow $f : \mathbf{Hd} \leftarrow \mathbf{Tl}$ of sort σ to the identity $id_{Hom(\mathbf{1},\sigma)}$, which is the unique arrow from $[\![\mathbf{Tl}]\!]_\sigma = Hom(\mathbf{1}, \sigma)$ to $[\![\mathbf{Hd}]\!]_\sigma = Hom(\mathbf{1}, \sigma)$. Intuitively, this means that every goal is true.

If P is atomic, we may define a much more interesting model, which maps a goal $\mathbf{G}$ to its set of correct ground answers:

$$[\![\mathbf{G}]\!]_\sigma = \{\mathsf{answer}(d) \mid d = \mathbf{G} \rightsquigarrow^* \top \text{ is a ground derivation}\} \ , \qquad (5.9)$$

where a *ground derivation* is a derivation whose last goal is in the fiber $\mathcal{P}(\mathbf{1})$. This assignment induces an unique interpretation. Moreover, for each clause $cl : \mathbf{Hd} \leftarrow \mathbf{Tl}$ of sort σ, if d is a ground derivation of $\mathbf{Tl}$, then $d' = \langle id_\sigma, cl \rangle \cdot d$ is a ground derivation for $\mathbf{Hd}$ with $\mathsf{answer}(d') = \mathsf{answer}(d)$. Therefore, $[\![\mathbf{Hd}]\!] \supseteq [\![\mathbf{Tl}]\!]$ and this gives an obvious mapping ι from clauses to arrows in the fibers of $\mathcal{G}_{\mathbb{C}}$. It turns out that $([\![\_]\!], \iota)$ is a model for P. ∎

When $\mathcal{P}$ 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 and preserves premonoidal structures. However, in the general case, other restrictions are imposed.

Example 5.21 Assume the hypotheses of Example 5.3. Consider the LP doctrine $\mathcal{G}_{\mathbb{C}}$ as defined in Example 5.20. An interpretation $[\![\_]\!]$ from $\mathcal{P}^{refp}_{\mathbb{J}}$ to $\mathcal{G}_{\mathbb{C}}$ is forced to map the arrow *refp* to an arrow in $\mathcal{G}_{\mathbb{C}}$. In symbols: $[\![p(\langle \pi_2, \pi_1 \rangle)]\!]_{\sigma\times\sigma} \supseteq [\![p(id)]\!]_{\sigma\times\sigma}$, i.e. $\langle \pi_2, \pi_1 \rangle^\sharp [\![p(id)]\!] \supseteq [\![p(id)]\!]$. This means that if $f \in [\![p(id)]\!]$ then $\langle \pi_2, \pi_1 \rangle \diamond f \in [\![p(id)]\!]$, i.e. $[\![p(id)]\!]$ is symmetric. ∎

Models and categorical derivations are strictly related, almost in the same way as proof systems and models are related in first order logic. A derivation $d : \mathbf{G} \rightsquigarrow^* \mathbf{G}'$, with $\mathsf{answer}(d) = t$, may be viewed as a proof that $t^\sharp\mathbf{G} \leftarrow \mathbf{G}'$, while the existence of an arrow $M(t^\sharp\mathbf{G}) \leftarrow M(\mathbf{G}')$ in a model M says that $t^\sharp\mathbf{G} \leftarrow \mathbf{G}'$ is true in the model M. Then, the following theorem essentially states that everything which is true is also provable and vice versa.

Theorem 5.22 (Soundness and completeness) *Given a program P over the LP doctrine $\mathcal{P}$, goals $\mathbf{G}' : \sigma$, $\mathbf{G} : \rho$ and an arrow $t : \sigma \to \rho$ in the base category of $\mathcal{P}$, the following conditions are equivalent:*

- *there is a categorical derivation $d : \mathbf{G} \rightsquigarrow^* \mathbf{G}'$ with answer $t : \sigma \to \rho$;*
- *in all models M of P, there is an arrow $M(t^\sharp\mathbf{G}) \leftarrow M(\mathbf{G}')$ in the fiber $M(\sigma)$.*

The proof of the theorem is long but straightforward. We follow the standard idea of building a "syntactical" model $F_P : \mathcal{P} \to \mathcal{F}_P$ of P starting from categorical derivations. Every arrow in the fiber σ of $\mathcal{F}_P$ corresponds to a *normal flat* derivation of sort σ, i.e., a derivation d such that

- for all steps $\langle r, f\rangle$ in d, it is the case that $r = id_\sigma$;
- there are no two consecutive steps with an arrow reduction pair;
- there are no steps with an arrow reduction pair $\langle r, f\rangle$ where f is an identity arrow.

Definition 5.23 (The LP doctrine $\mathcal{F}_P$) *Given a program P over $\mathcal{P}$, we define the LP doctrine $\mathcal{F}_P : \mathbb{C}^{\mathrm{o}} \to \mathsf{Cat}$ as follows:*

- *$\mathcal{F}_P(\sigma)$ is the category of normal flat derivations of sort σ. More in detail:*
  - *$|\mathcal{F}_P(\sigma)| = |\mathcal{P}\sigma|$;*
  - *$\mathcal{F}_P(\sigma)(\mathbf{G}, \mathbf{G}')$ is the set of normal flat derivations from $\mathbf{G}'$ to $\mathbf{G}$;*
  - *$id_{\mathbf{G}} = \epsilon_{\mathbf{G}}$;*
  - *$dd' = \mathsf{normalize}(d' \cdot d)$ where $\mathsf{normalize}$ collapses consecutive arrow reduction pair steps in order to get a normal derivation from $d' \cdot d$;*
- *given $t : \sigma \to \rho$ in $\mathbb{C}$, $\mathcal{F}_P(t)$ is defined as follows:*
  - *on objects, $\mathcal{F}_P(t)(\mathbf{G}) = t^\sharp\mathbf{G}$;*
  - *on arrows, $\mathcal{F}_P(t)(d) = \mathsf{normalize}(t^\sharp d)$, where $t^\sharp d$ is obtained by replacing each step $\langle id_\rho, f\rangle$ with $\langle id_\sigma, t^\sharp f\rangle$;*
- *for each $\sigma \in |\mathbb{C}|$, there is a strict premonoidal structure $(\otimes^{\mathcal{F}}_\sigma, \top^{\mathcal{F}}_\sigma)$ such that*
  - *$\top^{\mathcal{F}}_\sigma = \top^{\mathcal{P}}_\sigma$;*

– $\mathbf{G} \otimes^{\mathcal{F}}_{\sigma} d = \mathsf{normalize}(\mathbf{G} \otimes d)$ *and* $d \otimes^{\mathcal{F}}_{\sigma} \mathbf{G} = \mathsf{normalize}(d \otimes \mathbf{G})$*, where* $\mathbf{G} \otimes d$ *replaces each step* $\langle id_\sigma, f\rangle$ *with* $\langle id_\sigma, \mathbf{G} \otimes f\rangle$ *(analogously for* $d \otimes \mathbf{G}$*).*

The model $F_P : \mathcal{P} \to \mathcal{F}_P$ is obtained by mapping each arrow f in the fiber σ to the derivation $\langle id_\sigma, f\rangle$ and each clause $cl \in P$ of sort σ to $\langle id_\sigma, cl\rangle$. In formulas, $F_P = (\llbracket \_ \rrbracket, \iota)$ where $\llbracket \_ \rrbracket = \langle id_{\mathbb{C}}, \tau\rangle$, $\tau_\sigma(f) = \mathsf{normalize}(\langle id_\sigma, f\rangle)$ and $\iota(cl) = \langle id_\sigma, cl\rangle$.

The LP doctrine $\mathcal{F}_P$ may be described, without resorting to derivations, as the free indexed premonoidal category obtained from $\mathcal{P}$ by freely adjoining the clauses in P. This is formalized by the following universal mapping property:

Theorem 5.24 (Universal mapping property of F_P) *For each model* $M : \mathcal{P} \to \mathcal{Q}$ *for the program* P*, there exists an unique interpretation* $N : \mathcal{F}_P \to \mathcal{Q}$ *such that* $M = F_P \diamond N$*.*

In other words, $\mathcal{F}_P$ is the *category of proofs* induced by the primitive proofs in $\mathcal{P}$ and the added proofs given by clauses in P. The universal mapping property immediately gives soundness and completeness results. The detailed proof may be found in Section A.2.

5.4. Fixed Point Semantics

As promised, we now look for a fixed point semantic construction, similar in spirit to the characterization of least models as fixed points of the immediate consequence operator T_P of van Emden and Kowalski [79]. We start by examining the particular case of programs defined on the LP doctrine $\mathcal{P}_\Sigma$. Later, we will generalize the construction to work for different semantic and syntactic doctrines.

First of all, in the standard semantics for logic programs, the term *interpretation* is used to denote a subset I of all the ground atoms (the so called Herbrand base). This induces an interpretation in the style of Tarski, where term symbols denote term-building operators, and (overloading the symbol I), we define the following map from predicate symbols to sets of tuples of terms:

$$I(p) = \{\langle t_1, \ldots, t_n\rangle \mid p(t_1, \ldots, t_n) \in I\} \ . \tag{5.10}$$

As shown in Section 3, this corresponds to an indexed monoidal functor (which is our notion of interpretation), from $\mathcal{P}_\Sigma$ to the LP doctrine $\mathcal{G}_{\mathbb{S}_\sigma}$ (see Example 5.20), whose definition we restate for the reader's convenience:

- $\mathcal{G}_{\mathbb{S}_\sigma}(n)$ is the power set of ground substitutions $\{v_1/t_1, \ldots, v_n/t_n\}$;
- $\mathcal{G}_{\mathbb{S}_\sigma}(\gamma : m \to n)(X) = \{\theta \in \mathcal{G}_{\mathbb{S}_\sigma}(m) \mid \theta\gamma \in X\}$.

The fiber over n contains all the possible meanings for goals with n free variables. Reindexing along γ is the operator which returns the semantics of $p(\vec{t})\gamma$ from the semantics of $p(\vec{t})$. It is the preimage of composition of substitutions. We may build $\llbracket \_ \rrbracket$ from I as:

$$\llbracket p(v_1, \ldots, v_n)\rrbracket_n = \{\theta \in \mathcal{G}_{\mathbb{S}_\sigma}(n) \mid p(v_1, \ldots, v_n)\theta \in I\} \ , \tag{5.11}$$

$$[\![p(v_1,\ldots,v_n)\theta]\!]_m = \theta^\sharp\,[\![p(v_1,\ldots,v_n)]\!]_n \text{ for any } \theta : m \to n \ , \tag{5.12}$$

$$[\![p_1(\vec{t_1}),\ldots,p_n(\vec{t_n})]\!]_m = [\![p_1(\vec{t_1})]\!]_m \cap \ldots \cap [\![p_n(\vec{t_n})]\!]_m \ . \tag{5.13}$$

Since each atom $p(t_1,\ldots,t_n)$ is equal to $p(v_1,\ldots,v_n)\{v_1/t_n,\ldots,v_n/t_n\}$, the clauses above give semantics to every goal in $\mathcal{P}_\Sigma$.

Now, we come back to the problem of defining a fixed point semantics in the categorical framework. The standard T_P operator for logic programs is defined as

$$T_P(I) = \{A\theta \mid A \overset{cl}{\leftarrow} A_1,\ldots,A_n \in P, cl\theta \text{ is ground}, \{A_1\theta,\ldots,A_n\theta\} \subseteq I\} \ . \tag{5.14}$$

According to the above correspondence, we may rewrite T_P to work on indexed monoidal functors. We define $[\![\_]\!]' = T_P([\![\_]\!])$ as

$$[\![p(v_1,\ldots,v_n)]\!]'_n = \big\{\theta \diamond \{\vec{v}/\vec{t}\} \mid p(\vec{t}) \xleftarrow{cl:m} p_1(\vec{t_1}),\ldots,p_l(\vec{t_l}) \in P,\ \theta : \mathbf{1} \to m,\\ \theta \diamond \{\vec{v}/\vec{t_i}\} \in [\![p_i(v_1,\ldots,v_{k_i})]\!]_{k_i} \text{ for each } i \in \{1,\ldots,n\}\big\} \ , \tag{5.15}$$

where k_i is the arity of the predicate p_i and $\{\vec{v}/\vec{s}\} = \{v_1/s_1,\ldots,v_n/s_n\}$, provided that $\vec{s} = \langle s_1,\ldots,s_n\rangle$. The value of $[\![\_]\!]'$ for the other goals is completely determined by its value on pure atomic goals.

Interpretations from $\mathcal{P}_\Sigma$ to $\mathcal{G}_{\mathbb{S}_\sigma}$ are endowed with a partial order given by $[\![\_]\!] \leq [\![\_]\!]'$ iff $[\![\mathbf{G}]\!]_n \subseteq [\![\mathbf{G}]\!]'_n$ for each sort n and goal $\mathbf{G} : n$. T_P is continuous w.r.t. $\leq$, hence it has a least fixpoint which maps each goal to the set of its correct ground answers. This would be enough if we only wanted to give fixpoint semantics concerning ground answers of pure logic programs. However, this is hardly interesting, since we would just rewrite in a more complex form what has already been well-understood for decades [56]. In order to generalize this construction, we need to replace some operations, like set comprehension and substitution application, with something else with a more categorical flavor.

By properties of indexed premonoidal functors and definition of $\mathcal{G}_{\mathbb{S}_\sigma}$, we have that if $\theta : \mathbf{1} \to m$ then

$$\theta \diamond \{\vec{v}/\vec{t_i}\} \in [\![p_i(v_1,\ldots,v_{k_i})]\!]_{k_i} \text{ iff } \theta \in \{\vec{v}/\vec{t_i}\}^\sharp\,[\![p_i(v_1,\ldots,v_{k_i})]\!]_{k_i} = [\![p_i(\vec{t_i})]\!]_m \ .$$

This means that we may rewrite $[\![p(v_1,\ldots,v_n)]\!]'_n$ in (5.15) as

$$\begin{aligned}
&\{\theta \diamond \{\vec{v}/\vec{t}\} \mid p(\vec{t}) \xleftarrow{cl:m} \mathbf{G} \in P \wedge \theta \in [\![\mathbf{G}]\!]_m\}\\
=&\{\theta \diamond \gamma \mid \gamma : m \to n \wedge p(v_1,\ldots,v_n)\gamma \xleftarrow{cl:m} \mathbf{G} \in P \wedge \theta \in [\![\mathbf{G}]\!]_m\}\\
=&\bigcup\{\{\theta \diamond \gamma \mid \theta \in [\![\mathbf{G}]\!]_m\} \mid \gamma : m \to n \wedge p(v_1,\ldots,v_n)\gamma \xleftarrow{cl:m} \mathbf{G} \in P\}\\
=&\bigcup\{\Im_\gamma\,[\![\mathbf{G}]\!]_m \mid \gamma : m \to n \wedge p(v_1,\ldots,v_n)\gamma \xleftarrow{cl:m} \mathbf{G} \in P\}\\
=&\bigcup\{\Im_\gamma\,[\![\mathbf{G}]\!]_m \mid A \xleftarrow{cl:m} \mathbf{G} \in P \wedge \gamma^\sharp p(v_1,\ldots,v_n) = A\} \ ,
\end{aligned} \tag{5.16}$$

where $\Im_\gamma$ is the direct image of composition along γ. It is not difficult to check that $\Im_\gamma$ is the left adjoint to the reindexing map $\gamma^\sharp$.

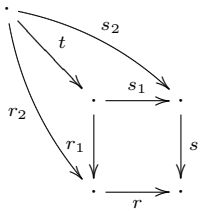
The idea is now to generalize (5.16) to work with other logic programming doctrines. However, while there are no problems in replacing $\mathcal{P}_\Sigma$ with a generic LP doctrine, the role of $\mathcal{G}_{\mathbb{S}_\sigma}$ may only be taken by doctrines which have at least coproducts (unions) and right adjoints to reindexing functors (direct images). Moreover, other properties will be required in order to ensure that the fixpoint operator is continuous. Therefore, we introduce the notion of semantic LP doctrine.

Definition 5.25 (Semantic LP doctrine) *A* semantic LP doctrine $\mathcal{Q}$ *is an LP doctrine where*

- *fibers are complete lattices;*
- *each premonoidal tensor $\otimes_\sigma$ is a join-complete functor (i.e. it is completely additive). In symbols: $(\bigvee_{i\in I} \mathbf{G}_i) \otimes_\sigma \mathbf{G} = \bigvee_{i\in I}(\mathbf{G}_i \otimes_\sigma \mathbf{G})$ and the same for the second argument of $\otimes_\sigma$;*
- *each reindexing functor $t^\sharp$ is join complete (i.e. it is completely additive);*
- *each reindexing functor $t^\sharp$ has a left-adjoint $\exists_t$;*
- (extended Beck-Chevalley condition) *given $r : \rho \to \alpha$, $s : \sigma \to \alpha$ and $X : \sigma$, then $r^\sharp \exists_s X = \bigvee\{\exists_{r_1} s_1{}^\sharp X \mid r_1 r = s_1 s\}$;*
- (Frobenius reciprocity) *given $r : \rho \to \sigma$, $X_1 : \rho$ and $X_2 : \sigma$, then $X_2 \otimes_\sigma \exists_r X_1 = \exists_r (r^\sharp X_2 \otimes_\rho X_1)$.*

We have chosen to restrict our study to fibers which are partial orders, instead of allowing more general categories, mainly with the aim of simplifying proofs and presentation.

Something should be said about the extended Beck-Chevalley condition. Note that, given the commutative diagram



(5.17)

we always[3] have $\exists_{r_1} s_1{}^\sharp X \leq r^\sharp \exists_s X$, and therefore $\vee\{\exists_{r'} s'{}^\sharp X \mid r'r = s's\} \leq r^\sharp \exists_s X$. The extended Beck-Chevalley condition only implies that this is actually an equality. Moreover, we may easily prove [4] that $\exists_{r_2} s_2{}^\sharp X \leq \exists_{r_1} s_1{}^\sharp X$. This means that, if the square s_1, r_1, s, r is a pullback diagram, then $\exists_{r_1} s_1{}^\sharp \geq \exists_{r'} s'{}^\sharp$

[3] From $s_1{}^\sharp s^\sharp = r_1{}^\sharp r^\sharp$, it follows that $\exists_{r_1} s_1{}^\sharp \leq \exists_{r_1} s_1{}^\sharp s^\sharp \exists_s = \exists_{r_1} r_1{}^\sharp r^\sharp \exists_s \leq r^\sharp \exists_s$.

[4] We have that $\exists_{r_2} s_2{}^\sharp = \exists_{r_1} \exists_t t^\sharp s_1{}^\sharp \leq \exists_{r_1} s_1{}^\sharp$.

for each r', s' such that $r'r = s's$, i.e. $\exists_{r_1} s_1{}^\sharp X = \vee\{\exists_{r'} s'^\sharp X \mid r'r = s's\}$. Hence, the extended Beck-Chevalley condition implies $\exists_{r_1} s_1{}^\sharp X = r^\sharp \exists_s X$, which is the standard *Beck-Chevalley condition* [46].

The Beck-Chevalley condition and Frobenius reciprocity always appear, in one form or another, in standard treatments of first-order categorical logic. Frobenius reciprocity is the categorical counterpart of the logical equivalence $\exists x.P \wedge Q \Leftrightarrow P \wedge \exists x.Q$ when x does not occur free in Q. The Beck-Chevalley condition is needed if want to give a logical interpretation to the object $\exists_t A$. It allows us to prove that $\exists_t A$ may be faithfully interpreted as $\exists y.(t(y) = x \wedge A(y))$ (see [78] for further details).

Example 5.26 (Ground answers) Given a finite product category $\mathbb{C}$, consider the indexed category $\mathcal{G}_\mathbb{C}$ as defined in Example 5.20. It is possible to turn $\mathcal{G}_\mathbb{C}$ into a semantic LP doctrine. Actually:

- Each fiber is a complete distributive lattice.
- $\otimes$ is the meet of the lattice, hence it is additive.
- $\mathcal{G}_\mathbb{C}(t)$ is a complete join-morphism for each t.
- We can define $\exists_t$, with $t : \sigma \to \rho$ as the function which maps an $X \subseteq Hom_\mathbb{C}(\mathbf{1}, \sigma)$ to
$$\{rt \mid r \in X\} \ ,$$
which is a subset of $Hom_\mathbb{C}(\mathbf{1}, \rho)$. $\exists_t$ is the left adjoint of $\mathcal{G}_\mathbb{C}(t)$, since
$$(\mathcal{G}_\mathbb{C}(t) \diamond \exists_t)(X) = \{r \in X \mid r \text{ factors trough } t\} \subseteq X \ ,$$
$$(\exists_t \diamond \mathcal{G}_\mathbb{C}(t))(X) = \{r \mid \exists r' \in X.\, r't = rt\} \supseteq X \ .$$
- The extended Beck-Chevalley condition holds. Given $t \in r^\sharp \exists_s X$, we have that $t \diamond r \in \exists_s X$, hence $tr = t's$ for some $t' \in X$. Therefore $id_\mathbf{1} \in t'^\sharp X$ and $t \in \exists_t t'^\sharp X$.
- The Frobenius reciprocity holds. Given $r : \rho \to \sigma$, $X_2 \subseteq Hom(\mathbf{1}, \sigma)$ and $X_1 \subseteq Hom(\mathbf{1}, \rho)$, we have $\exists_r(r^\sharp X_2 \cap X_1) = \exists_r\{t \mid tr \in X_2 \wedge t \in X_1\} = \{tr \mid tr \in X_2 \wedge t \in X_1\} = \{t' \mid t' \in X_2 \wedge t' = tr \wedge t \in X_1\} = X_2 \cap \exists_r X_1$.

■

Since fibers in the semantic LP doctrines are posets, the notion of model is simplified.

Proposition 5.27 *If $\mathcal{Q}$ is a semantic LP doctrine and $[\![\_]\!] : \mathcal{P} \to \mathcal{Q}$ is an interpretation, then $[\![\_]\!]$ may be extended to a model for a program P iff for each clause $cl : \mathbf{Hd} \leftarrow \mathbf{Tl}$, we have $[\![\mathbf{Hd}]\!] \geq [\![\mathbf{Tl}]\!]$.*

Proof. Let $\iota(cl)$ be the unique arrow from $[\![\mathbf{Tl}]\!]$ to $[\![\mathbf{Hd}]\!]$, where uniqueness comes from the fact that the fiber is a partial order. Then $M = ([\![\_]\!], \iota)$ is the only model which extends $[\![\_]\!]$. On the other side, if $([\![\_]\!], \iota)$ is a model, then $\iota(cl : \mathbf{Hd} \leftarrow \mathbf{Tl}) : [\![\mathbf{Hd}]\!] \leftarrow [\![\mathbf{Tl}]\!]$, i.e. $[\![\mathbf{Hd}]\!] \geq [\![\mathbf{Tl}]\!]$. ■

Since the extension is unique, in the following we will just say that $[\![\_]\!]$ is a model when it satisfies the condition of Proposition 5.27.

Interpretations from $\mathcal{P}$ to $\mathcal{Q}$ with a fixed change of base functor F are endowed with a straightforward pointwise partial order. Namely,

$$\langle F, \tau\rangle \leq \langle F, \tau'\rangle \iff \forall \mathbf{G} : \sigma.\, \tau_\sigma(\mathbf{G}) \leq \tau'_\sigma(\mathbf{G}) \ . \tag{5.18}$$

Given a collection $\{[\![\_]\!]_i\}_{i\in I}$ of interpretations, its least upper bound may be defined as:

$$(\vee_{i\in I}\, [\![\_]\!])_\sigma(\mathbf{G}) = \vee_{i\in I}\, [\![\mathbf{G}]\!]_\sigma \ . \tag{5.19}$$

Now, assume we have an interpretation $[\![\_]\!] = \langle F, \tau\rangle$ from $\mathcal{P}$ to $\mathcal{Q}$, where $\mathcal{Q}$ is a semantic LP doctrine. We want to incrementally build a new interpretation which is also a model of $\mathcal{P}$. At the end, we want to define an operator $\mathbb{E}_P$ on interpretations such that the colimit of the chain $\mathbb{E}^i_P([\![\_]\!])$ for $i \in \mathbb{N}$ may be extended to a model for P. $\mathbb{E}_P$ essentially works by using clauses to augment the interpretation $[\![\_]\!]$. If there is a clause $cl : \mathbf{Hd} \leftarrow \mathbf{Tl}$, then $\mathbb{E}_P([\![\_]\!])(\mathbf{Hd})$ should contain both the original semantics of $\mathbf{Hd}$, namely $[\![\mathbf{Hd}]\!]$, and the semantics of $\mathbf{Tl}$, namely $[\![\mathbf{Tl}]\!]$. However, we need to be careful if we want to ensure $\mathbb{E}_P([\![\_]\!])$ is actually an interpretation. In particular, in order to guarantee that $\mathbb{E}_P([\![\_]\!])$ preserves composition of arrows, we introduce the concept of *reducer*.

Definition 5.28 (Reducers) *A* reducer *of the goal* $\mathbf{G} : \sigma$ *into* $\mathbf{Tl} : \rho$ *is a triple* $\langle r, f, fcl\rangle$ *where* $r : \rho \to \sigma$, $fcl : \mathbf{Hd} \leftarrow \mathbf{Tl}$ *is a formal clause of type* ρ *and* $f : r^\sharp\mathbf{G} \leftarrow \mathbf{Hd}$ *is an arrow in* $\mathcal{P}\rho$. *Note that* $\langle r, f\rangle$ *is a reduction pair for* $\mathbf{G}$.

A reducer $\langle r, f, fcl\rangle$ corresponds to the categorical derivation $\langle r, f\rangle \cdot \langle id, fcl\rangle$. $\mathbb{E}_P$ implicitly builds categorical derivations bottom-up, using reducers as the basic building blocks.

Example 5.29 Consider the factorial program in Example 4.1. Let $\mathbf{G}$ be the goal $\mathsf{fact}(\langle 4, id_{n_1}\rangle)$ of sort n_1, (corresponding to the query $\mathtt{fact(4,A)}$). If $r = \times\langle 4, id_{n_1}\rangle$ (corresponding to the substitution $\{\mathtt{A/4*A}\}$), $f = id_{\mathbf{G}}$ and $fcl = \langle \top_{n_1}, \langle 3, id_{n_1}\rangle, cl2, \top_{n_1}\rangle$ (corresponding to $\mathtt{fact(4,4*A)} :- \mathtt{fact(3,A)}$, an instance of the second clause), then $\langle r, f, fcl\rangle$ is a reducer of $\mathbf{G}$ into $\mathsf{fact}(\langle 3, id_{n_1}\rangle)$ (corresponding to $\mathtt{fact(3,A)}$). Actually, it is the case that $r \diamond \langle 4, id_{n_1}\rangle = \langle 4, \times\langle 4, id_{n_1}\rangle\rangle = \langle 3, id_{n_1}\rangle \diamond \langle +\langle \mathsf{l}, 1\rangle, \times\langle +\langle \mathsf{l}, 1\rangle, \mathsf{r}\rangle\rangle$. ■

In the case of non-atomic logic programs, we have the same problem we already encountered for standard semantics such as ground correct answers. The existence of derivations of $\mathbf{G}_1 \otimes \mathbf{G}_2$ which are not related to derivations of $\mathbf{G}_1$ and $\mathbf{G}_2$, makes it difficult, if not impossible, to define a valid immediate consequence operator. In particular, it seems that $\mathbb{E}_P$ may be forced to either preserve the premonoidal structure, or to preserve arrows between interpretations, but not both. Therefore, we restrict attention to atomic programs, where all these problems disappear. Whenever the operator $\mathbb{E}_P$ is used, we always implicitly assume that P is atomic.

Here and in the following, given an interpretation $[\![\_]\!] = \langle F, \tau\rangle : \mathcal{P} \to \mathcal{Q}$, we will write $\exists_t$ as a short form of $\exists^{\mathcal{Q}}_{Ft}$.

Definition 5.30 (Successive consequences operator $\mathbb{E}_P$) *Given an interpretation $[\![\_]\!]$, we define $\mathbb{E}_P([\![\_]\!]) = \langle F, \tau' \rangle$ according to the following equations:*

$$\begin{aligned}
&\tau'_\sigma(A) = [\![A]\!]_\sigma \vee \bigvee \{\exists_r [\![\mathbf{Tl}]\!]) \mid \langle r, f, fcl \rangle \text{ is a reducer of } A \text{ into } \mathbf{Tl}\} \ , \\
&\tau'_\sigma(\top^{\mathcal{P}}_\sigma) = \top^{\mathcal{Q}}_{F\sigma} \ , \\
&\tau'_\sigma(A_1 \otimes_\sigma \cdots \otimes_\sigma A_n) = \tau'_\sigma(A_1) \otimes^{\mathcal{Q}}_{F\sigma} \cdots \otimes^{\mathcal{Q}}_{F\sigma} \tau'_\sigma(A_n) \ ,
\end{aligned} \tag{5.20}$$

where A, $A_1, \ldots, A_n$ denote atomic goals. Since the fibers in $\mathcal{Q}$ are partial orders, the definition of τ' on arrows is forced by its definition on objects.

Note that the first equation in the previous definition is essentially the generalization of (5.15), while the other two equations are generalizations of (5.12) and (5.13) respectively. The two main differences are: 1) $\tau'_\sigma(A)$ always includes $\tau_\sigma(A)$, which means that $\mathbb{E}_P$ is extensive (i.e., $\mathbb{E}_P([\![\_]\!]) \geq [\![\_]\!]$); 2) we use reducers to deal with arrows in $\mathcal{P}$, in particular in order to ensure that τ' may be extended to arrows. The latter is explained in detail in the following Proposition.

Proposition 5.31 *If $[\![\_]\!] : \mathcal{P} \to \mathcal{Q}$ is an interpretation, then $\mathbb{E}_P([\![\_]\!])$ is an interpretation.*

Proof. If $\mathbb{E}_P([\![\_]\!]) = \langle F, \tau' \rangle$, we need to check that

- τ'_σ may be defined on arrows, i.e. if there is an arrow $\mathbf{G}_1 \to \mathbf{G}_2$ in $\mathcal{P}\sigma$, then $\tau'_\sigma(\mathbf{G}_1) \subseteq \tau'_\sigma(\mathbf{G}_2)$;
- τ' is natural in σ, i.e. for each arrow $s : \rho \to \sigma$, $s^\flat \tau'_\sigma(\mathbf{G}) = \tau'_\rho(s^\sharp \mathbf{G})$;
- τ' preserves the premonoidal structure.

First of all, we prove that τ' preserves the premonoidal structure. Obviously it maps identities to identities, hence we only need to prove that, given $\mathbf{G} = \mathbf{G}_1 \otimes \mathbf{G}_2$, we have $\tau'(\mathbf{G}) = \tau'(\mathbf{G}_1) \otimes \tau'(\mathbf{G}_2)$. If $\mathbf{G}$ is atomic, then either $\mathbf{G}_1 = \top$ or $\mathbf{G}_2 = \top$, and the property follows immediately. If $\mathbf{G} = \otimes^n_{i=1} A_i$ then $\mathbf{G}_1 = \otimes^j_{i=1} A_i$ and $\mathbf{G}_2 = \otimes^n_{i=j+1} A_i$, hence $\tau'(\mathbf{G}) = \otimes^n_{i=1} \tau'(A_i) = (\otimes^j_{i=1} \tau'(A_i)) \otimes (\otimes^n_{i=j+1} \tau'(A_i)) = \tau'(\mathbf{G}_1) \otimes \tau'(\mathbf{G}_2)$ by associativity of $\otimes$.

Now we come back to the first property. Assume there is an arrow $f : \mathbf{G}_1 \to \mathbf{G}_2$ over the fiber σ. The proof is by induction on the number of atoms n in the decomposition of $\mathbf{G}_2$. If $n = 0$ then $\mathbf{G}_1 = \mathbf{G}_2 = \top_\sigma$, hence trivially $\tau'_\sigma(\top) \leq \tau'_\sigma(\top)$. If $n = 1$ then $\mathbf{G}_2$ is an atom. Since $[\![\_]\!]$ is an interpretation, then $[\![\mathbf{G}_1]\!] \leq [\![\mathbf{G}_2]\!]$. If $\langle r, f', fcl \rangle$ is a reducer of $\mathbf{G}_1$ into $\mathbf{Tl}$, then $fcl : \mathbf{Hd} \leftarrow \mathbf{Tl}$ and $f' : \mathbf{Hd} \to r^\sharp \mathbf{G}_1$. If we define $f'' = f' \diamond r^\sharp f$, we have that $f'' : \mathbf{Hd} \to r^\sharp \mathbf{G}_2$. Therefore $\langle r, f'', fcl \rangle$ is a reducer of $\mathbf{G}_2$ into $\mathbf{Tl}$. Hence, all the components of $\tau'(\mathbf{G}_1)$ in (5.20) also appear in $\tau'(\mathbf{G}_2)$. If $n > 1$ then $\mathbf{G}_2 = \otimes^n_{i=1} A_i$ and $\tau'(\mathbf{G}_2) = \otimes^n_{i=1} \tau'(A_i)$. Since $\mathcal{P}$ is atomic, there are arrows $f_i : \mathbf{G}'_i \to A_i$ such that $\mathbf{G}_1 = \mathbf{G}'_1 \otimes \cdots \otimes \mathbf{G}'_n$. By inductive hypothesis, $\tau'(A_i) \geq \tau'(\mathbf{G}'_i)$, hence $\tau'(\mathbf{G}_2) = \otimes^n_{i=1} \tau'(A_i) \geq \otimes^n_{i=1} \tau'(\mathbf{G}'_i) = \tau'(\mathbf{G}_1)$.

Now we check the second condition. If $\mathbf{G} = A$ is an atom, since $s^\flat$ is additive and $[\![\_]\!]$ is an interpretation,

$$s^\flat \tau'_\sigma(A) = \left[\!\left[s^\sharp A\right]\!\right]_\rho \vee \bigvee \{ s^\flat \exists_r [\![\mathbf{Tl}]\!]_{\sigma'} \mid \langle r, f, fcl\rangle \text{ is a reducer of } A : \sigma \text{ into } \mathbf{Tl} : \sigma' \} \tag{5.21}$$

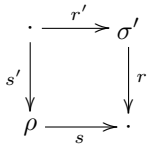
while

$$\tau'_\rho(s^\sharp A) = \left[\!\left[s^\sharp A\right]\!\right]_\rho \vee \bigvee \{ \exists_r [\![\mathbf{Tl}]\!]_{\sigma'} \mid \langle r, f, fcl\rangle \text{ reducer of } s^\sharp A : \rho \text{ into } \mathbf{Tl} : \sigma' \} \ . \tag{5.22}$$

Consider one of the components $s^\flat \exists_r [\![\mathbf{Tl}]\!]_{\sigma'}$ in (5.21). By the extended Beck condition we know that

$$\begin{aligned} s^\flat \exists_r [\![\mathbf{Tl}]\!]_{\sigma'} &= \bigvee \{ \exists_{s'} r'^\flat [\![\mathbf{Tl}]\!]_{\sigma'} \mid r'r = s's \} \\ &= \bigvee \{ \exists_{s'} \left[\!\left[r'^\sharp \mathbf{Tl} \right]\!\right]_{\mathsf{dom}(r')} \mid r'r = s's \} \ . \end{aligned} \tag{5.23}$$

The relation between r, r', s and s' is pictured below



Note that, since $\langle r, f, fcl\rangle$ is a reducer of A into $\mathbf{Tl}$, then $f : r^\sharp A \leftarrow \mathbf{Hd}$ and $fcl : \mathbf{Hd} \leftarrow \mathbf{Tl}$. If $r'r = s's$ then $r'^\sharp r^\sharp A = s'^\sharp s^\sharp A$ and therefore $r'^\sharp(f) : s'^\sharp s^\sharp A \leftarrow r'^\sharp \mathbf{Hd}$. This means that $\langle s', r'^\sharp(f), r'^\sharp fcl\rangle$ is a reducer of $s^\sharp A : \rho$ into $r'^\sharp \mathbf{Tl}$, and therefore (5.23) is included in (5.22).

For the opposite containment, assume $\langle r, f, fcl\rangle$ is a reducer of $s^\sharp A$ into $\mathbf{Tl}$. This means that $\langle rs, f, fcl\rangle$ is a reducer of A into $\mathbf{Tl}$. Therefore, $s^\flat \tau'_\sigma(A) \geq s^\flat \exists_{r \diamond s} [\![\mathbf{Tl}]\!]_{\sigma'} = s^\flat \exists_s \exists_r [\![\mathbf{Tl}]\!]_{\sigma'} \geq \exists_r [\![\mathbf{Tl}]\!]_{\sigma'}$. This proves that (5.22) is included in (5.21).

If $\mathbf{G} = \top$, the property follows since $\tau'(\top_\sigma) = \top_{F\sigma}$ for each σ and since reindexing functors preserve the identity elements. If $\mathbf{G} = A_1, \ldots, A_n$, then $s^\flat \tau'_\sigma(\mathbf{G}) = s^\flat(\tau'_\sigma(A_1) \otimes_{F\sigma} \cdots \otimes_{F\sigma} \tau'_\sigma(A_n)) = s^\flat \tau'_\sigma(A_1) \otimes_{F\rho} \cdots \otimes_{F\rho} s^\flat \tau'_\sigma(A_n) = \tau'_\rho(s^\sharp(A_1) \otimes_\rho \cdots \otimes_\rho s^\sharp(A_n)) = \tau'_\rho(s^\sharp \mathbf{G})$, since reindexing functors preserve the premonoidal structures. ■

Given an interpretation $[\![\_]\!]$, we are also interested in computing the least fixed point of $\mathbb{E}_P$ greater than $[\![\_]\!]$. We need to be sure that such an object exists. Hence, we need to prove that $\mathbb{E}_P$ is extensive and monotone. Actually, a stronger property may be proved.

Proposition 5.32 *$\mathbb{E}_P$ is completely additive and extensive.*

Proof. Additivity immediately follows by additivity of $\exists_f$ and $\otimes$. Extensivity follows by extensivity on atoms. ■

The reason we are interested in the fixed points of $\mathbb{E}_P$ is their connection to models of P, as the following theorem clarifies.

Theorem 5.33 *An interpretation $[\![\_]\!] : \mathcal{P} \to \mathcal{Q}$ is a model for the program P iff it is a fixed point of $\mathbb{E}_P$.*

Proof. We will separately prove the two implications. Let us assume that $[\![\_]\!]$ is a fixed point of $\mathbb{E}_P$. This means that, for each clause $\mathbf{Hd} \xleftarrow{cl:\sigma} \mathbf{Tl}$, $[\![\mathbf{Hd}]\!] \geq [\![\mathbf{Tl}]\!]$, since $\langle id_\sigma, id_{\mathbf{Hd}}, cl\rangle$ is a reducer of $\mathbf{Hd}$ into $\mathbf{Tl}$. By Proposition 5.27, $[\![\_]\!]$ is a model.

Now assume $[\![\_]\!]$ is a model. Consider an atomic goal A and a reducer $\langle r, f, fcl\rangle$ of A into $\mathbf{Tl}$ such that $fcl = \langle \mathbf{G}_1, s, cl, \mathbf{G}_2\rangle$ and $cl : \mathbf{Hd}' \leftarrow \mathbf{Tl}'$. Hence $fcl : \mathbf{G}_1 \otimes s^\sharp\mathbf{Hd}' \otimes \mathbf{G}_2 \leftarrow \mathbf{G}_1 \otimes s^\sharp\mathbf{Tl}' \otimes \mathbf{G}_2$ and $f : r^\sharp A \leftarrow \mathbf{G}_1 \otimes s^\sharp\mathbf{Hd}' \otimes \mathbf{G}_2$. Since $[\![\_]\!]$ is a model, then $[\![\mathbf{Hd}']\!] \geq [\![\mathbf{Tl}']\!]$. Since both monoidal tensor and reindexing are monotone (by functoriality), then $[\![\mathbf{G}_1 \otimes s^\sharp\mathbf{Hd}' \otimes \mathbf{G}_2]\!] \geq [\![\mathbf{G}_1 \otimes s^\sharp\mathbf{Tl}' \otimes \mathbf{G}_2]\!] = [\![\mathbf{Tl}]\!]$, while the existence of the arrow f implies $[\![r^\sharp A]\!] \geq [\![\mathbf{G}_1 \otimes s^\sharp\mathbf{Hd}' \otimes \mathbf{G}_2]\!] \geq [\![\mathbf{Tl}]\!]$. Therefore $[\![A]\!] \geq \exists_r r^\flat [\![A]\!] = \exists_r [\![r^\sharp A]\!] \geq \exists_r [\![\mathbf{Tl}]\!]$. Thus $\mathbb{E}_P([\![\_]\!])(A) = [\![A]\!]$. This equality immediately extends to $\mathbb{E}_P([\![\_]\!])(\mathbf{G}) = [\![\mathbf{G}]\!]$ for every goal $\mathbf{G}$, since interpretations of goals uniquely depend from the interpretations of atomic goals. ∎

We have now all the pieces to build a model starting from an interpretation $[\![\_]\!]$. We repeatedly apply $\mathbb{E}_P$ to $[\![\_]\!]$, and take the least upper bound of all the interpretations we obtain. Since $\mathbb{E}_P$ is additive and extensive, the least upper bound is the least fixed point of $\mathbb{E}_P$ greater than $[\![\_]\!]$.

Theorem 5.34 *Given a program P over $\mathcal{P}$, a semantic LP doctrine $\mathcal{Q}$ and an interpretation $[\![\_]\!] : \mathcal{P} \to \mathcal{Q}$, $\mathbb{E}_P^\omega([\![\_]\!])$ is the least model of P greater than $[\![\_]\!]$.*

Proof. It is a standard result of lattice theory. ∎

To ease notation, in the following we will write $\mathbb{E}_P^n([\![\_]\!])$ as $[\![\_]\!]^n$. In particular, this means $[\![\_]\!]^0 = [\![\_]\!]$ and $[\![\_]\!]^\omega = \mathbb{E}_P^\omega([\![\_]\!])$.

Remark 5.35 (Free-goal programs) Let $\mathcal{P} : \mathbb{C}^{\mathrm{o}} \to \mathsf{Cat}$ be an atomic LP doctrine with the following additional properties:

- there exists a family of atoms $\{A_i : \sigma_i\}_{i\in I}$ such that, for each atom A, there is an unique choice of $i \in I$ and $t \in \mathbb{C}$ such that $A = t^\sharp A_i$;
- the only arrows in the fibers are the identities.

This is a very special notion of LP doctrine, which we will call *free-goal LP doctrine*. An atomic program P over $\mathcal{P}$ is called a *free-goal program* when $\mathcal{P}$ is free-goal. A free-goal program is the counterpart in our framework of programs as defined in [24], i.e. programs which do not admit logical proofs at the level of predicates. If we work with free-goal LP doctrines, we will denote a goal $t^\sharp A_i$ also as $A_i(t)$, in order to stress the similarities with standard logic programs.

If P is free-goal, then $\mathbb{E}_P$ may be rewritten in a simpler form. Namely, $\mathbb{E}_P(\langle F, \tau\rangle) = \langle F, \tau'\rangle$ with

$$\begin{aligned}
&\tau'_{\sigma_i}(A_i) = [\![A_i]\!]_{\sigma_i} \vee \bigvee\{\exists_r [\![\mathbf{Tl}]\!] \mid cl : A_i(r) \leftarrow \mathbf{Tl} \in P\} \ , \\
&\tau'_\sigma(\top^{\mathcal{P}}_\sigma) = \top^{\mathcal{Q}}_{F\sigma} \ , \\
&\tau'_\sigma(A_i(r)) = r^\flat(\tau'_{\sigma_i}(A_i)) \text{ if } r : \sigma \to \sigma_i \ , \\
&\tau'_\sigma(A_1(r_1), \ldots, A_n(r_n)) = \tau'_\sigma(A_1(r_1)) \otimes^{\mathcal{Q}}_{F\sigma} \cdots \otimes^{\mathcal{Q}}_{F\sigma} \tau'_\sigma(A_n(r_n)) \ .
\end{aligned} \tag{5.24}$$

This fixed point construction was actually the only one provided in a previous version of this work [2, 1]. This is more similar to the standard T_P operator than (5.20), since it essentially computes the semantics only for pure atomic goals (the A_i's), while other goals are obtained by reindexing.

Note that the syntactic doctrines $\mathcal{P}_\Sigma$ and $\mathcal{P}_{\mathbb{J}}$ are free-goal. ■

Example 5.36 If we write the definition of $[\![\_]\!]' = \mathbb{E}_P([\![\_]\!])$ in all the details for the syntactic doctrine $\mathcal{P}_\Sigma$ in the Example 5.26 and the semantic doctrine $\mathcal{G}_{\mathbb{S}_\Sigma}$, following Remark 5.35 we have that:

$$\begin{aligned}
&[\![A_i]\!]'_{\sigma_i} = [\![A_i]\!]_{\sigma_i} \cup \bigcup\{tr \mid t \in [\![\mathbf{Tl}]\!], cl : A_i(r) \overset{cl}{\leftarrow} \mathbf{Tl} \in P\} \ , \\
&[\![A_i(r)]\!]'_\sigma = \{t \mid tr \in [\![A_i]\!]_{\sigma_i}\} \text{ if } r : \sigma \to \sigma_i \ , \\
&[\![A_1(r_1), \ldots, A_n(r_n)]\!]'_\sigma = [\![A_1(r_1)]\!]_\sigma \cap \ldots \cap [\![A_n(r_n)]\!]_\sigma \ .
\end{aligned} \tag{5.25}$$

If we work within the LP doctrine $\mathcal{P}_\Sigma$, then $\mathbb{E}_P([\![\_]\!])$ becomes equivalent to T_P semantics for pure logic programs given by (5.16) and (5.15), with the only exception that the inclusion of $[\![A_i]\!]_{\sigma_i}$ in $[\![A_i]\!]'_{\sigma_i}$ makes $\mathbb{E}_P$ extensive.

In the example which follows, assume to work in the LP doctrine $\mathcal{P}_{\mathbb{J}}$ with base category Set and a single predicate symbol $p : \mathbb{N}$. The program P is composed of two clauses $p(\mathsf{succ} \diamond \mathsf{succ}) \overset{cl_1}{\longleftarrow} p(id_{\mathbb{N}})$ and $p(0) \overset{cl_2}{\longleftarrow} \top$. Let $[\![\_]\!]$ be the unique interpretation which maps p to $\emptyset$. Then, we may compute successive steps of the $\mathbb{E}_P$ operator starting from $[\![\_]\!]$. We obtain $[\![p(id_{\mathbb{N}})]\!]^n = \{0, 2, \ldots, 2(n-1)\}$, where a number i denotes the arrow $f : \mathbf{1} \to \mathbb{N}$ which picks out i. If we take the upper bound of this chain, we get $[\![p(id_{\mathbb{N}})]\!]^\omega = \{2n \mid n \in \mathbb{N}\}$, which is what we would expect from the intuitive meaning of the program P. ■

Before we consider the problem of completeness of fixed point semantics, we want to briefly come back to the problem of non-atomic programs. We think that, in order to give a good fixed point semantics for them, the semantic doctrine should be endowed with left adjoints to the tensor products. Actually, it is $\exists_t$, the left adjoint to reindexing, which allows us to compute the semantics of the goal $\mathbf{G}$ using the clauses whose head is $t^\sharp\mathbf{G}$. If $\exists_t$ did not exist, we would be forced to only consider clauses whose head were a pure atomic goal. Therefore, we think that having a left adjoint of $\otimes$, would allow us to determine the effect of a clause $\mathbf{G}_1 \otimes \mathbf{G}_2 \leftarrow \mathbf{Tl}$ with respect to the goal $\mathbf{G}_1$, in such a way that, later, the semantics of $\mathbf{G}_1 \otimes \mathbf{G}_2$ may be derived uniquely by the semantics of $\mathbf{G}_1$ and $\mathbf{G}_2$. We leave the details needed to exploit this idea to future work.

5.5. Completeness of Fixed Point Semantics

In the previous section we have given a fixed point construction that builds a model of a program given a starting interpretation for the syntactic doctrine. What is the relationship between this model and the operational semantics?

First of all, consider that, in general, we do not want a full completeness result. This means that, if $[\![\_]\!]$ is the fixed point semantics and there is an arrow from $[\![\mathbf{G}]\!]$ to $[\![\mathbf{G}']\!]$, we do not expect to find an SLD derivation from $\mathbf{G}'$ to $\mathbf{G}$. For example, consider the following pure logic program in the LP doctrine $\mathcal{P}_\Sigma$ for an appropriate signature Σ:

$$\mathsf{div2}(0) \leftarrow \top \qquad \mathsf{div4}(0)$$
$$\mathsf{div2}(s(s(v_1))) \leftarrow \mathsf{div2}(v_1) \qquad \mathsf{div4}(s(s(s(s(v_1))))) \leftarrow \mathsf{div4}(v_1)$$

With the ground answer semantic doctrine introduced in the Example 5.26, $\mathbb{E}_P$ becomes equivalent to the standard T_P operator. If $[\![\_]\!]$ is the least fixpoint of $\mathbb{E}_P$, $[\![\mathsf{div2}(v_1))]\!]_1$ and $[\![\mathsf{div4}(v_1)]\!]_1$ are the set of ground answers for the goals $\mathsf{div2}(v_1)$ and $\mathsf{div4}(v_1)$ respectively. Since the fibers in the semantic doctrine are partial orders, there is an arrow $[\![\mathsf{div4}(v_1)]\!]_1 \to [\![\mathsf{div2}(v_1)]\!]_1$. However, there is no SLD derivation of the kind $\mathsf{div2}(v_1) \overset{\theta}{\rightsquigarrow}{}^* \mathsf{div4}(v_1)$, and we did not expect any.

However, assume we have an arrow from $\top$ to $[\![\mathbf{G}]\!]$. Under appropriate conditions, we would expect to find a successful derivation for the goal $\mathbf{G}$.

Definition 5.37 (Weak completeness) *Given a program P over $\mathcal{P}$ and an interpretation $[\![\_]\!] = \langle F, \tau\rangle : \mathcal{P} \to \mathcal{Q}$, we say $[\![\_]\!]$ is* weakly complete *when, for each arrow $\top^{\mathcal{Q}}_{F\sigma} \to [\![\mathbf{G}]\!]_\sigma$, there is an SLD derivation $d : \mathbf{G} \rightsquigarrow^* \top^{\mathcal{P}}_\sigma$ with $\mathsf{answer}(d) = id_\sigma$.*

We would like to prove that $[\![\_]\!]^\omega$ is weakly complete when $[\![\_]\!]$ is weakly complete. We require additional properties on the semantic doctrine.

Definition 5.38 (Well-behaved units) *A semantic LP doctrine $\mathcal{Q}$ has* well-behaved units *when the following properties hold:*

1. ***(coprimality)** if $\top_\sigma \leq \bigvee_{i\in I} X_i$ in the fiber $\mathcal{Q}\sigma$, there exists $j \in I$ such that $\top_\sigma \leq X_j$;*
2. *if $t : \sigma \to \rho$ and $\top_\rho \leq \exists_t X$, there is $s : \rho \to \sigma$ such that $st = id_\rho$ and $\top_\rho \leq s^\sharp X$.*

Although these conditions may appear rather obscure, they have a precise meaning from the logical point of view: they correspond to the disjunctive and existential properties of intuitionistic logic.

Example 5.39 Consider the ground answers semantic doctrine $\mathcal{G}_\mathbb{C}$ in Example 5.26. In general, units in $\mathcal{G}_\mathbb{C}$ are not well-behaved. Assume $\mathbb{C}$ is the full subcategory of Set made of the sets $\mathbb{N}^n$ for all $n \in \mathbb{N}$. In the fiber $\mathbb{N}$, let $A = \{\lambda x : \mathbf{1}.\, n \mid n = 1\} \subseteq \mathit{Hom}(\mathbf{1}, \mathbb{N})$ and $B = \{\lambda x : \mathbf{1}.\, n \mid n > 1\} \subseteq \mathit{Hom}(\mathbf{1}, \mathbb{N})$. Then $\top_\mathbb{N} = \mathit{Hom}(\mathbf{1}, \mathbb{N}) = A \vee B$, but neither $\top_\mathbb{N} \subseteq A$ nor $\top_\mathbb{N} \subseteq B$. We will see later an example of a semantic doctrine with well-behaved units. ∎

Theorem 5.40 *Given a weakly complete interpretation $[\![\_]\!]$ of an LP doctrine $\mathcal{P}$ in a semantic LP doctrine with well-behaved units $\mathcal{Q}$ and a program P over $\mathcal{P}$, then $[\![\mathbf{G}]\!]^\omega$ is weakly complete.*

Proof. The proof of this theorem may be found in the Appendix. ∎

6. Yoneda Semantics

We have shown it is possible to define several kinds of fixed point semantics for our programs, according to the chosen semantic doctrine. Now we will focus our attention on the *Yoneda semantics*, so called since it is related to the Yoneda embedding. We will show it has particularly strong completeness properties w.r.t. declarative semantics.

Note that it is possible to define a notion of canonical subobject in $\mathsf{Set}^{\mathbb{C}^o}$, where $H \overset{m}{\longmapsto} F$ is canonical when m_σ is set inclusion for each $\sigma \in |\mathbb{C}|$. In particular, a canonical subobject of $Hom(\_, \sigma)$ in $\mathsf{Set}^{\mathbb{C}^o}$ can be thought of as a *sieve:* left-closed set of arrows targeted at σ. We will often use implicitly such a correspondence.

Definition 6.1 (Yoneda doctrine) *Given a category $\mathbb{C}$, the* Yoneda doctrine *over $\mathbb{C}$ is the premonoidal indexed category $\mathcal{Y}_\mathbb{C} : \mathbb{C}^o \to \mathsf{Cat}$, such that*

- *$\mathcal{Y}_\mathbb{C}(\sigma)$ is the complete lattice of canonical subobjects of $Hom(\_, \sigma)$;*
- *$\mathcal{Y}_\mathbb{C}(r : \rho \to \sigma)$ takes the left-closed span $X \subseteq Hom(\_, \sigma)$ to the set of arrows $\{t \in \mathbb{C} \mid tr \in X\}$;*
- *for each $\sigma \in \mathbb{C}$, the strict premonoidal structure $(\otimes_\sigma, \top_\sigma)$ is given by defining $\top_\sigma = Hom(\_, \sigma)$ and $X_1 \otimes_\sigma X_2 = X_1 \cap X_2$.*

Theorem 6.2 *The doctrine $\mathcal{Y}_\mathbb{C}$ is a semantic LP doctrine.*

Proof. It is similar to the proof that $\mathcal{G}_\mathbb{C}$ is a semantic LP doctrine. Full details may be found in the Appendix. ∎

Given an LP doctrine $\mathcal{P}$ on the base category $\mathbb{C}$, consider $[\![\_]\!]^* = \langle id_\mathbb{C}, \tau \rangle : \mathcal{P} \to \mathcal{Y}_\mathbb{C}$ such that

$$[\![\mathbf{G}]\!]^*_\sigma = \{r : \rho \to \sigma \mid \text{ there exists } r^\sharp \mathbf{G} \leftarrow \top_\rho \text{ in } \mathcal{P}(\rho)\} \ . \tag{6.1}$$

It is the case that $\mathbf{G}$ is true according to the model $[\![\_]\!]^*$ (i.e., $[\![\mathbf{G}]\!]^*_\sigma = \top_\sigma$) when it has a proof which only involves a priori information built-in on the syntactic doctrine, without even specifying a program.

Proposition 6.3 *If $\mathcal{P}$ is atomic, $[\![\_]\!]^*$ is a weakly complete interpretation.*

Proof. We first show that it is an indexed functor:

1. Given an arrow $g : \mathbf{G}_1 \to \mathbf{G}_2 \in \mathcal{P}\sigma$, we need to prove $[\![\mathbf{G}_2]\!]^*_\sigma \supseteq [\![\mathbf{G}_1]\!]^*_\sigma$. If $r : \rho \to \sigma \in [\![\mathbf{G}_1]\!]^*_\sigma$, it means there exists an arrow $f : r^\sharp \mathbf{G}_1 \leftarrow \top_\rho$ in $\mathcal{P}\rho$. Hence, $f \diamond r^\sharp(g) : r^\sharp \mathbf{G}_2 \leftarrow \top_\rho$ is an arrow in $\mathcal{P}\rho$, hence $r \in [\![\mathbf{G}_2]\!]^*_\sigma$.
2. Given an arrow $t : \alpha \to \sigma$ in $\mathbb{C}$, we prove that $t^\flat\, [\![\mathbf{G}]\!]^*_\sigma = \left[\!\left[t^\sharp \mathbf{G}\right]\!\right]^*_\alpha$. In fact $\left[\!\left[t^\sharp \mathbf{G}\right]\!\right]^*_\alpha = \{r : \rho \to \alpha \mid r^\sharp t^\sharp \mathbf{G} \leftarrow \top_\alpha \in \mathcal{P}\alpha\} = \{r : \rho \to \alpha \mid rt \in [\![\mathbf{G}]\!]^*_\sigma\} = t^\flat\, [\![\mathbf{G}]\!]^*_\sigma$.

Moreover $[\![\_]\!]^*$ preserves the unit of the premonoidal structure: $[\![\top_\sigma]\!]^* = \{r : \rho \to \sigma \mid r^\sharp \top_\sigma \leftarrow \top_\rho \in \mathcal{P}\rho\} = \{r : \rho \to \sigma \mid \top_\rho \leftarrow \top_\rho \in \mathcal{P}\rho\} = \mathit{Hom}(\_, \sigma)$.

If $\mathcal{P}$ is atomic, $[\![\_]\!]^*$ also preserves the premonoidal tensor. We have that $[\![\mathbf{G}_1 \otimes_\sigma \mathbf{G}_2]\!]^*_\sigma = \{r : \rho \to \sigma \mid r^\sharp \mathbf{G}_1 \otimes_\rho r^\sharp \mathbf{G}_2 \leftarrow \top_\rho\}$. Since $\mathcal{P}$ is atomic, if $f : r^\sharp \mathbf{G}_1 \otimes_\rho r^\sharp \mathbf{G}_2 \leftarrow \top_\rho$, we may assume without loss of generality there are $f_1 : r^\sharp \mathbf{G}_1 \leftarrow \top_\rho$ and $f_2 : r^\sharp \mathbf{G}_2 \leftarrow \top_\rho$ such that $f = (\top_\rho \otimes f_2) \diamond (f_1 \otimes r^\sharp \mathbf{G}_2)$. Therefore $r \in [\![\mathbf{G}_1]\!]^*_\sigma \cap [\![\mathbf{G}_2]\!]^*_\sigma$. For the converse, if $r : \rho \to \sigma \in [\![\mathbf{G}_1]\!]^*_\sigma \cap [\![\mathbf{G}_2]\!]^*_\sigma$, there are arrows $f_1 : r^\sharp \mathbf{G}_1 \leftarrow \top_\rho$ and $f_2 : r^\sharp \mathbf{G}_2 \leftarrow \top_\rho$, therefore there is an arrow $(f_1 \otimes \top_\rho) \diamond (r^\sharp \mathbf{G}_1 \otimes f_2) : r^\sharp(\mathbf{G}_1 \otimes \mathbf{G}_2) \leftarrow \top_\rho$. This proves $r \in [\![\mathbf{G}_1 \otimes \mathbf{G}_2]\!]^*_\sigma$.

Now we prove it is weakly complete. Assume $\mathit{Hom}(\_, \sigma) \subseteq [\![\mathbf{G}]\!]^*$. This means $id_\sigma \in [\![\mathbf{G}]\!]^*$, hence there is an arrow $f : \mathbf{G} \leftarrow \top_\sigma \in \mathcal{P}\sigma$. Therefore $\langle id_\sigma, f\rangle$ is a reduction pair of $\mathbf{G}$ into $\top_\sigma$ and $d : \mathbf{G} \xrightarrow{\langle id, f\rangle} \top_\sigma$ is a successful derivation of $\mathbf{G}$ with answer id_σ. ∎

We apply the results of Theorem 5.34 to obtain the so called *Yoneda model* of P, namely $Y_P = \mathbb{E}^\omega_P([\![\_]\!]^*)$. Since $[\![\_]\!]^*$ is weakly complete, we only need to prove that $\mathcal{Y}_\mathbb{C}$ has well-behaved units, in order to show that Y_P is a weakly complete model.

Theorem 6.4 *The semantic doctrine $\mathcal{Y}_\mathbb{C}$ has well-behaved units.*

Proof. Let us start with coprimality. If $\top_\sigma \subseteq \bigcup_{j\in J} X_j$, there is $i \in J$ such that $id_\sigma \in X_i$. Since X_i is left-closed, then $\top_\sigma \subseteq X_i$.

Now, assume $\top_\rho \subseteq \exists_t X$. This means there is $s : \rho \to \sigma \in X$ such that $st = id_\rho$. By definition of $s^\sharp$ we also have $s^\sharp X = id_\rho$ and since objects are left-closed, we have $\top_\rho \subseteq s^\sharp X$. ∎

Note that Y_P has strong completeness properties w.r.t. categorical derivations, which are not direct consequences of Theorem 5.34. In particular, we may prove that $Y_P(\mathbf{G})$ is the set of correct answers for the goal $\mathbf{G}$.

Theorem 6.5 *For every LP doctrine $\mathcal{P}$ and program P over $\mathcal{P}$, we have that*

$$Y_P(\mathbf{G}) = \{r \mid r \textit{ is a correct answer for } \mathbf{G}\} \ .$$

Proof. Since $Y_P(\mathbf{G})$ is left-closed, it is obvious that $r \in Y_P(\mathbf{G})$ iff $r^\flat Y_P(\mathbf{G}) = Y_P(r^\sharp \mathbf{G}) = \mathit{Hom}(\_, \rho) = \top_\rho$. By Theorem 5.40, the model Y_P is weakly complete, hence $Y_P(r^\sharp \mathbf{G}) = \top_\rho$ iff there is a derivation $d = r^\sharp \mathbf{G} \rightsquigarrow^* \top_\rho$, with $\mathsf{answer}(d) = id_\rho$.

Now, note that it is possible to define an operator $\mathsf{flatten}$ (see the Appendix for more details) such that, given a derivation $d : \mathbf{G} \rightsquigarrow^* \mathbf{G}'$ with $\mathsf{answer}(d) =$

$r : \rho \to \sigma$, $\mathsf{flatten}(d)$ is a derivation $d' = r^\sharp \mathbf{G} \rightsquigarrow^* \mathbf{G}'$ with empty answer. On the converse, given a derivation $d' = r^\sharp \mathbf{G} \rightsquigarrow^* \mathbf{G}'$ with $\mathsf{answer}(d') = id_\rho$ and $r : \rho \to \sigma$, we have that $\langle r, id_\rho \rangle \cdot d$ is a derivation of $d : \mathbf{G} \rightsquigarrow^* \mathbf{G}'$ with answer r. Therefore r is a correct answer for $\mathbf{G}$ iff id is a correct answer for $r^\sharp \mathbf{G}$. This concludes the proof. ∎

Remark 6.6 (Abstract interpretation) In [15] a theory of observables is presented, using abstract interpretation [19, 20] to relate different semantics for logic programs. The same idea may be applied to our framework. Although a detailed discussion of abstraction in categorical logic programming is beyond the scope of the paper, we want to show an example which relates two of the semantics we have introduced so far. Namely, the Yoneda semantics and the ground answer semantics.

Consider a finite product category $\mathbb{C}$, and the two semantic LP doctrines $\mathcal{G}_\mathbb{C}$ and $\mathcal{Y}_\mathbb{C}$. We may define a pair of adjoint indexed functors $A = \langle id_\mathbb{C}, \alpha \rangle : \mathcal{Y}_\mathbb{C} \to \mathcal{G}_\mathbb{C}$ and $\Gamma = \langle id_\mathbb{C}, \gamma \rangle : \mathcal{G}_\mathbb{C} \to \mathcal{Y}_\mathbb{C}$ as follows:

$$\alpha_\sigma(X) = X \cap Hom(\mathbf{1}, \sigma) \qquad \gamma_\sigma(X) = \{t : \rho \to \sigma \mid \forall t' : \mathbf{1} \to \rho,\ t't \in X\} \ . \quad (6.2)$$

The model $Y_P \diamond A$ is the semantics of correct ground answers for the program P. It maps the $\mathbf{G} : \sigma$ to the set $\{r : \mathbf{1} \to \mathbf{G} \mid \ r$ is a correct ground answer for $\mathbf{G}\}$. Moreover, the $\mathbb{E}_P$ operator for ground answer semantics $\left(\mathbb{E}_P^{\mathcal{G}_\mathbb{C}}\right)$ may be obtained from the $\mathbb{E}_P$ operator for Yoneda semantics $\left(\mathbb{E}_P^{\mathcal{Y}_\mathbb{C}}\right)$, as

$$\mathbb{E}_P^{\mathcal{G}_\mathbb{C}}([\![\_]\!]) = \mathbb{E}_P^{\mathcal{Y}}([\![\_]\!] \diamond \Gamma) \diamond A \ . \quad (6.3)$$

We may prove that $(\mathbb{E}_P^{\mathcal{G}_\mathbb{C}})^\omega([\![\_]\!]^* \diamond A) = Y_P \diamond A$. This states that the ground answer semantics $Y_P \diamond A$ may be computed alternatively as the least fixpoint of $\mathbb{E}_P^{\mathcal{G}_\mathbb{C}}$, starting from the interpretation $[\![\_]\!]^* \diamond A$.

Adjunctions may be used to relate Y_P with other semantics of logic programs or with semantic domains used for static analysis, such as groundness [18] or sharing [39]. In the latter case, if A is the abstraction which maps the Yoneda doctrine $\mathcal{Y}_\mathbb{C}$ to the semantic doctrine $\mathcal{Q}$ of the observable property of interest, it is possible to prove that $(\mathbb{E}_P^{\mathcal{Q}})^\omega([\![\_]\!]^* \diamond A) \geq Y_P \diamond A$. In other words, a fixpoint computation within $\mathcal{Q}$ gives a correct approximation of the real property of interest. ∎

7. An Example: Constraint Logic Programming

One of the most common extensions to pure logic programming is *constraint logic programming* (CLP) [40]. In CLP, terms and built-in predicates are not interpreted in the Herbrand universe but in a fixed algebra, which depends on the particular CLP language in use. CLP($\mathcal{R}$) [41] is the particular instance of CLP where the symbols `+`, `-`, `*`, `/` `<=` and the numeric literals `0`, `1`, ... are interpreted over the algebra of real numbers. This means that, for example, the goal `X+1 <= X+2` is vacuously true. The following is a program in CLP($\mathcal{R}$) to compute the factorial of a natural number:

```
fact(0,1).
fact(N,N*Y) :- N > 0, fact(N-1, Y).
```

This kind of improvement comes for free when we adopt our categorical framework: it is enough to choose as base category the algebraic category of interest. For example, for CLP($\mathcal{R}$) we could work in the base category given by the Lawvere Algebraic Theory for the real numbers (see Example 4.1 for the analogous construction for natural numbers).

However, CLP is more than just computing in a particular algebraic domain. It is also about computing with partial information, since constraints, and not just values, may be input, or returned by a computation. In CLP($\mathcal{R}$), for example, an answer for the goal $\mathtt{p(X)}$ may not be representable by a finite set of substitutions like $\{\mathtt{X/4}\}$ or $\{\mathtt{X/Y*Z}\}$. Answers may be constraints, such as $\mathtt{X} > \mathtt{Y}$ which has an infinite number of solutions, or $\mathtt{X*X*X} \approx \mathtt{6}$ (using $\_ \approx \_$ to represent equality constraints, as distinct from equality in the metalanguage) which cannot be represented in the form $\{\mathtt{X/t}\}$, since the cube root is a not a symbol of the signature.

Abstracting from the particular CLP language in use, a constraint logic program is a set of clauses of the form

$$p(\vec{t}) \leftarrow c, p_1(\vec{t_1}), \ldots, p_n(\vec{t_n}) \ ,$$

where c is a constraint in an appropriate *constraint system*. Sometimes $\square$ is used to separate the constraint from the rest of the body in the clause. Execution proceeds as for standard logic programming, but the constraints in the clauses are accumulated, and possibly modified in a *constraint store*, and then returned as the result of the computation. If the constraint store reach an inconsistent state, computation is halted, and failure is returned.

Example 7.1 Consider the following circuit example in CLP($\mathcal{R}$) adapted by [40]:

```
circuit(resistor(R), V, I) :- V = R I.
circuit(diode(V, 10 V + 1000) :- V < -100.
circuit(diode(V, 0.001 V) :- -100 ≤ V, V ≤ 0.6.
circuit(diode(V, 100 V - 60) :- V > 0.6.
circuit(series(N₁, N₂), V, I) :- V = V₁ + V₂, circuit(N₁, V₁, I),
                                 circuit(N₂, V₂, I).
```

The goal $\mathbf{G} = \mathtt{R} \geq \mathtt{0}, \mathtt{circuit(series(resistor(R), diode), 5, I)}$ is immediately rewritten into $\mathbf{G}' = \mathtt{circuit(resistor(R), V_1, I), circuit(diode, V_2, I)}$ with the constraint store $\mathtt{R} \geq \mathtt{0}, \mathtt{V_1} + \mathtt{V_2} = \mathtt{5}$. Later $\mathbf{G}'$ is rewritten into $\mathbf{G}'' = \mathtt{circuit(diode, V_2, I)}$ with the constraint $\mathtt{R} \geq \mathtt{0}, \mathtt{V_1} + \mathtt{V_2} = \mathtt{5}, \mathtt{V_1} = \mathtt{RI}$. Using the first clause for diodes, we reach the empty goal with the constraint store $\mathtt{R} \geq \mathtt{0}, \mathtt{V_1} + \mathtt{V_2} = \mathtt{5}, \mathtt{V_1} = \mathtt{RI}, \mathtt{I} = \mathtt{10V_2} + \mathtt{1000}, \mathtt{V_2} < -\mathtt{100}$. This set of equations and inequations is inconsistent, hence the CLP interpreter backtracks and tries to solve $\mathbf{G}'$ using the second clauses for diodes, obtaining the empty goal with

$\mathtt{R} \geq 0, \mathtt{V_1} + \mathtt{V_2} = 5, \mathtt{V_1} = \mathtt{RI}, \mathtt{I} = 0.001\mathtt{V_2}, -100 \leq \mathtt{V_2} \leq 0.6$, which may be simplified into $0 \leq \mathtt{R} \leq 7333.\bar{3}, \mathtt{V_1} + \mathtt{V_2} = 5, \mathtt{V_1} = \mathtt{RI}, \mathtt{I} = 5/(1000 + \mathtt{R})$. Later, computation proceeds by looking for a solution using the third clause for diodes. ■

We will show that CLP may be viewed as a simple instance of our framework. First of all, note that the constraint store plays the role of local logical state for CLP systems, hence it is a perfect candidate for embedding in the base category.

In the doctrine $\mathcal{P}_\Sigma$ (see Example 5.2), which is the categorical counterpart of pure logic programming, states (objects in the base category) represent sets of allowed free variables and state transitions are substitutions, which change the current set of free variables and instantiate the goals in the fibers. Now we also add constraints in the base category, whose objects become pairs $\langle n, c\rangle$ where n is the number of free variables allowed in the goal and c is the constraint. The idea is that goals and arrows which live above $\langle n, c\rangle$ in a fiber depend on the constraint in the corresponding base object. For example, the two goals $p(v_1 * v_1)$ and $p(2)$ should be isomorphic if the constraint in the base entails $v_1 * v_1 \approx 2$.

In this way, constraints need never appear explicitly in the goals: they are just "types" which annotate goals and clauses. How can we encode the clause $p(\vec{t}) \leftarrow c, p_1(\vec{t_1}), \ldots, p_n(\vec{t_n})$ if c does not appear explicitly in the fibers? The idea is that, from the logical point of view

$$p(\vec{t}) \leftarrow c, p_1(\vec{t_1}), \ldots, p_n(\vec{t_n}) \iff (p(\vec{t}) \leftarrow p_1(\vec{t_1}), \ldots, p_n(\vec{t_n})) \leftarrow c \ . \qquad (7.1)$$

Therefore c may be factored out: when c holds, the entire clause is true. The CLP clause (7.1) becomes a clause $p(\vec{t}) \leftarrow p_1(\vec{t_1}), \ldots, p_n(\vec{t_n})$ above $\langle m, c\rangle$, where m is the number of free variables.

An arrow θ from $\langle m, c\rangle$ to $\langle m', c'\rangle$ is an arrow θ in $\mathbb{S}_\Sigma$, with the additional condition that c implies $c'\theta$. For example, given $\rho = \langle 1, v_1 * v_1 \approx 2\rangle$ and $\sigma = \langle 1, v_1 > 0\rangle$, then $\theta = \{v_1/v_1 * v_1\}$ is an arrow from ρ to σ since $v_1 * v_1 \approx 2$ implies $(v_1 > 0)\theta$, i.e. $v_1 * v_1 > 0$.

According to the definition of categorical derivation, if we want to solve the goal $p(\vec{t}) : \langle m', c'\rangle$ with the clause $cl : \langle m, c\rangle$ given in (7.1), we need to find two arrows $\theta_1 : \langle l, c''\rangle \to \langle m', c'\rangle$ and $\theta_2 : \langle l, c''\rangle \to \langle m, c\rangle$ such that $p(\vec{t}\theta_1) = p(\vec{t'}\theta_2)$. This means we need to find a unifier of $\vec{t}$ and $\vec{t'}$ but, at the same time, we need to move to a fiber where $c\theta_2$ is true. In other words, we need to add c (or one of its instances) to the constraint store.

7.1. General Results

We now present a theory which realizes and generalizes the ideas we introduced above. It is evident that we need a categorical counterpart of a *constraint system*. We use the definition in [68].

Definition 7.2 (Constraint system) *A* constraint system *over the category $\mathbb{C}$ with finite products is an indexed category $\mathcal{D}$ over $\mathbb{C}$ such that each fiber is a bounded meet preorder (i.e. a preorder with terminal elements and finite products) and reindexing functors have left adjoints. We denote by $r^\sharp$ the reindexing*

functor $\mathcal{D}(r)$ and with $\exists_r$ its left adjoint. We also require the following two conditions:

- ***Beck-Chevalley condition**: If the following diagram is a pullback*

$$\begin{array}{ccc} \cdot & \xrightarrow{s_1} & \sigma \\ {\scriptstyle r_1}\downarrow & & \downarrow{\scriptstyle s} \\ \rho & \xrightarrow[r]{} & \alpha \end{array}$$

  and $c \in \mathcal{D}(\sigma)$, then $\exists_{r_1} {s_1}^\sharp c = r^\sharp \exists_s c$.

- ***Frobenius reciprocity**: For each $r : \rho \to \sigma$ in $\mathbb{C}$, $c_1 \in \mathcal{D}(\rho)$ and $c_2 \in \mathcal{D}(\sigma)$, we have $c_2 \wedge \exists_r c_1 = \exists_r (r^\sharp c_2 \wedge c_1)$.*

The idea is the same we use in our framework: the base category represents terms and types, and for each type we have a fiber which contains all the constraints over that type. Reindexing is instantiation of constraints, while left adjoints to reindexing functors represent the existential quantification of constraints.

Example 7.3 (The constraint system $\mathcal{R}$) Let $\mathbb{C}$ be the full subcategory of Set made of all the objects of the form $\mathbb{R}^n$ for $n \in \mathbb{N}$. We define a constraint system $\mathcal{R}$ for real numbers as follows. For each $n \in \mathbb{N}$, the fiber over $\mathbb{R}^n$, is the power set $\wp(\mathbb{R}^n)$. For $n = 0$, we let $\mathbb{R}^0$ be a set with only one element.

Given $X \subseteq \mathbb{R}^n$, reindexing along the function $t : \mathbb{R}^m \to \mathbb{R}^n$ is given by the preimage, i.e. $t^\sharp X = t^{-1}(X)$, while the right adjoint is given by direct image, i.e. $\exists_t X = t(X)$. It is easy to check that $\mathcal{R}$ defined in this way is a constraint system. The proof may be found in Proposition A.12.

Traditionally, constraints for the language of real numbers are given as formulas in some fragment of first order logic. There is a direct correspondence between a constraint in syntactic form and in the form presented here. For example, the constraint $x \leq y$ becomes $\{(u, v) \in \mathbb{R}^2 \mid u \leq v\}$. Obviously, we may express constraints which do not have a syntactic counterpart, at least with the standard set of term and predicate symbols.

We could stick more strictly to the practice by defining a syntactical variant of $\mathcal{R}$, where the base category is the Lawvere Algebraic Theory for the real numbers, and fibers over n are given by formulas with n free variables built from a fixed set of predicate symbols over reals (such as "=" and "<"). However, giving the full details of this construction would be quite long, hence we prefer the semantic approach which has a straightforward presentation. ■

Let us call true_σ the greatest element in the fiber σ. We will drop σ from the subscript when it is not relevant or when it is clear from the context. For each $\sigma \in |\mathbb{C}|$, there is a diagonal arrow $\Delta_\sigma = \langle id_\sigma, id_\sigma \rangle$ and $\exists_{\Delta_\sigma} \mathsf{true}_\sigma$ of type $\sigma \times \sigma$ is a constraint which behaves like $x \approx y$ where x, y are variables over σ (proofs may be found in [78]). Therefore, although not explicitly stated, all our

constraint systems have equality: given $t_1 : \rho \to \sigma$ and $t_2 : \rho \to \sigma$, consider the following diagram on the base category:

$$\rho \xrightarrow{\langle t_1, t_2 \rangle} \sigma \times \sigma \xleftarrow{\Delta_\sigma} \sigma \quad .$$

We write $t_1 \approx t_2$ as a short form for $\langle t_1, t_2 \rangle^\sharp(\exists_{\Delta_\sigma} \mathsf{true}_\sigma)$, which is the constraint enforcing equality of the terms t_1 and t_2.

Example 7.4 Consider the constraint system $\mathcal{R}$ shown above, and two arrows $t_1, t_2 : \mathbb{R} \to \mathbb{R}$ such that $t_1(x) = 2x$ and $t_2(x) = x + 2$. The map $\Delta_\mathbb{R} : \mathbb{R} \to \mathbb{R}^2$ is $\Delta_\mathbb{R}(x) = \langle x, x \rangle$, while $\mathsf{true}_\mathbb{R} = \mathbb{R}$. Therefore $\langle t_1, t_2 \rangle^\sharp(\exists_{\Delta_\mathbb{R}} \mathsf{true}_\mathbb{R}) = \langle t_1, t_2 \rangle^\sharp\{(x, x) \mid x \in \mathbb{R}\} = \{y \mid (t_1(y), t_2(y)) \in \{(x, x) \mid x \in \mathbb{R}\}\} = \{y \mid t_1(y) = t_2(y)\} = \{y \mid 2y = y + 2\} = \{2\}$, which is exactly what we would expect by the semantics of the equality constraint $t_1 \approx t_2$, i.e. the set of values v such that $t_1(v) = t_2(v)$. ■

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 [37]. To be more precise

Definition 7.5 (Grothendieck construction) *Given a constraint system $\mathcal{D}$ over $\mathbb{C}$, we denote by $\mathbb{D}$ the category obtained as follows:*

- *objects of $\mathbb{D}$ are pairs $\langle \sigma, c \rangle$ where $\sigma \in |\mathbb{C}|$ and $c \in |\mathcal{D}(\sigma)|$;*
- *arrows in $\mathbb{D}$ from $\langle \sigma_1, c_1 \rangle$ to $\langle \sigma_2, c_2 \rangle$ are given by arrows $t : \sigma_1 \to \sigma_2$ in $\mathbb{C}$ such that $c_1 \leq t^\sharp c_2$.*

The category $\mathbb{D}$ will be the base category for our syntactic doctrine $\mathcal{P}$. For each pair $\langle \sigma, c \rangle$, the fiber $\mathcal{P}(\langle \sigma, c \rangle)$ will contain the goals of type σ. These goals should be well-behaved w.r.t. the constraint c. In the standard setting this means that $\mathtt{p(X * X)}$ and $\mathtt{p(2)}$ should be isomorphic when c implies $\mathtt{X * X} \approx \mathtt{2}$. Otherwise, if the square root is not in the signature of the language, the goal $\mathtt{X * X} \approx \mathtt{2}, \mathtt{p(X * X)}$ has no successful derivation, even in the presence of the clause $\mathtt{p(2)}$. In the general settings, the following regularity condition should hold:

$$\forall \mathbf{G} : \langle \sigma, c \rangle,\ t_1 : \langle \rho, c' \rangle \to \langle \sigma, c \rangle,\ t_2 : \langle \rho, c' \rangle \to \langle \sigma, c \rangle,$$
$$c' \leq t_1 \approx t_2 \Rightarrow {t_1}^\sharp(\mathbf{G}) \text{ and } {t_2}^\sharp(\mathbf{G}) \text{ are isomorphic.} \quad (7.2)$$

If (7.2) does not hold, nothing prevent us from applying the results of Section 5, but we argue that the resulting language cannot be considered an instance of CLP, since constraints and reindexing do not agree. Since there is no prior presentation of CLP in so much general terms, we cannot made this statement more precise.

Definition 7.6 *A* CLP *program over the constraint system $\mathcal{D}$ is a logic program P over $\mathcal{P}$ such that the base category of $\mathcal{P}$ is $\mathbb{D}$, the fibration corresponding to $\mathcal{D}$ by Grothendieck construction, and the fibers respect the regularity condition in* (7.2)*.*

To our CLP programs we may apply all the results in Section 5 about models and derivations. In particular, note that given a goal $\mathbf{G}$ in the fiber $\langle \sigma, \mathsf{true} \rangle$ and a derivation $d : \mathbf{G} \rightsquigarrow^* \top$, then $\mathsf{answer}(d) : \langle \rho, c \rangle \xrightarrow{r} \langle \sigma, \mathsf{true} \rangle$ contains both an answer term r and an answer constraint c for the goal $\mathbf{G}$. We may also apply the $\mathbb{E}_P$ operator to build least fixed point semantics for atomic programs.

7.1.1. Inconsistent states

In general, constraint systems also have an inconsistent state false_σ for each fiber σ, which is the bottom of the meet-semilattice. In this case, we require false to be preserved by reindexing functors. This means that, once we reach a state $\langle \sigma, \mathsf{false}_\sigma \rangle$ during a derivation, we are doomed to remain in a state with an inconsistent constraint. Generally, CLP interpreters stop the current derivation when they reach an inconsistent state, but from the logical point of view, there is nothing wrong in continuing.

On the contrary, if we want the behavior of our programs to be closer to standard CLP languages, we may change the way we build $\mathbb{D}$ from $\mathcal{D}$. The second point in Definition 7.5 is replaced by:

- arrows in $\mathbb{D}$ from $\langle \sigma_1, c_1 \rangle$ to $\langle \sigma_2, c_2 \rangle$ are given by arrows $t : \sigma_1 \to \sigma_2$ in $\mathbb{C}$ such that $c_1 \leq t^\sharp c_2$ **and** $\boldsymbol{c_1 \not\leq \mathsf{false}_{\sigma_1}}$**.**

In this way, inconsistent constraints may never be reached during execution of the program.

Another way to handle inconsistency of constraints is to use the standard Grothendieck construction, but to add the following complementary regularity condition:

$$\forall \sigma \in \mathbb{C}, \text{ all objects in } \mathcal{P}(\langle \sigma, \mathsf{false}_\sigma \rangle) \text{ are isomorphic.} \tag{7.3}$$

Intuitively, this means that, in an inconsistent state, everything is true. Therefore, upon reaching the goal $\mathbf{G}$ in an inconsistent state $\langle \sigma, \mathsf{false}_\sigma \rangle$, we may obtain a successful derivation with the single step $\langle id_{\langle \sigma, \mathsf{false}_\sigma \rangle}, i \rangle$ where $i : \top \to \mathbf{G}$ is the appropriate isomorphism.

7.2. Logic Programs with Constraints

As an example of CLP language, we now present a simple extension of $\mathcal{P}_\mathbb{J}$ which incorporates constraints. If $\mathcal{D}$ is the constraint system, the new LP doctrine will be called $\mathcal{P}_{\mathbb{J},\mathcal{D}}$. Atomic programs over $\mathcal{P}_{\mathbb{J},\mathcal{D}}$ correspond to standard (syntactic) CLP programs in the constraint system $\mathcal{D}$. This correspondence is not only syntactical, but extends to derivations in both realms, hence to models and fixed point operators, too.

Let $\mathcal{D}$ be a constraint system over $\mathbb{C}$ and $\mathbb{J}$ a discrete display category, endowed with a functor $\delta : \mathbb{J} \to \mathbb{C}$. We define the atomic LP doctrine $\mathcal{P}_{\mathbb{J},\mathcal{D}}$, similar to $\mathcal{P}_\mathbb{J}$ but with the addition of constraints:

1. the base category is $\mathbb{D}$, obtained from $\mathcal{D}$ via the Grothendieck construction;

2. $\mathcal{P}_{\mathbb{J},\mathcal{D}}(\langle\sigma, c\rangle)$ is the discrete category whose objects are possibly empty sequences of *atomic goals*. An atomic goal is an equivalence class of pairs $\langle p, t\rangle$ such that $p \in |\mathbb{J}|$ and $t : \sigma \to \delta(p)$ is an arrow in $\mathbb{C}$, modulo the following equivalence relation:

$$\langle p, t_1\rangle \asymp \langle p, t_2\rangle \iff c \leq t_1 \approx t_2 \ . \tag{7.4}$$

   To ease notation, we write $p(t)$ instead of $[\langle p, t\rangle]_{\asymp}$;
3. $\mathcal{P}_{\mathbb{J},\mathcal{D}}(r)$ where $r : \langle\rho, c'\rangle \to \langle\sigma, c\rangle$ maps the goal $p_1(t_1), \ldots, p_n(t_n) \in |\mathcal{P}_{\mathbb{J}}(\langle\sigma, c\rangle)|$ to $p_1(rt_1), \ldots, p_n(rt_n)$;
4. the premonoidal structure in $\mathcal{P}_{\mathbb{J},\mathcal{D}}(\langle\sigma, c\rangle)$ is $(\otimes_{\sigma,c}, \top_{\sigma,c})$ where $\otimes_{\sigma,c}$ is given by concatenation of sequences and $\top_{\sigma,c}$ is the empty sequence. It is actually a monoidal structure.

Note that $\asymp$ is an equivalence relation, thanks to the properties of $\approx$. Moreover, reindexing functors are well-defined. Assume $r : \langle\rho, c'\rangle \to \langle\sigma, c\rangle$. If $\langle p, t_1\rangle \asymp \langle p, t_2\rangle$ in $\langle\sigma, c\rangle$, then $c \leq t_1 \approx t_2$ in $\mathcal{D}(\sigma)$, hence $c' \leq r^\sharp(c) \leq r^\sharp(t_1 \approx t_2) = (rt_1 \approx rt_2)$ in $\mathcal{D}(\rho)$. Finally, Equation 7.2 holds, since given the atomic goal $\langle p, t\rangle_{\asymp}$ over the sort $\langle\sigma, c\rangle$ and two arrows $r_1, r_2 : \langle\rho, c'\rangle \to \langle\sigma, c\rangle$, we have that ${r_1}^\sharp\langle p, t\rangle_{\asymp} = \langle p, r_1 t\rangle_{\asymp}$ and ${r_2}^\sharp\langle p, t\rangle_{\asymp} = \langle p, r_2 t\rangle_{\asymp}$. If $c' \leq r_1 \approx r_2$ then $c' \leq r_1 t \approx r_2 t$ for any t such that $\mathsf{dom}(t) = \mathsf{cod}(r_1)$ (see proof in the Appendix), hence $\langle p, r_1 t\rangle_{\asymp}$ and $\langle p, r_2 t\rangle_{\asymp}$ are isomorphic (actually equal).

In the general case, $\mathcal{P}_{\mathbb{J},\mathcal{D}}$ is an atomic LP doctrine but it is not free-goal. For example, let $\mathbb{J}$ be the discrete category with a single object p and $\delta(p) = \mathbb{R}$. Moreover, let us consider the constraint system $\mathcal{R}$ from Example 7.3. The only sensible choice of pure atoms is the singleton $\{A\}$, where $A = \langle p, id_{\mathbb{R}}\rangle : \langle\mathbb{R}, \mathsf{true}_{\mathbb{R}}\rangle$. Actually, every atomic goal may be obtained by reindexing A, but not in an unique way. Given the arrows $t = id_{\mathbb{R}} : \langle\mathbb{R}, 2 \approx id_{\mathbb{R}}\rangle \to \langle\mathbb{R}, \mathsf{true}_{\mathbb{R}}\rangle$ and $r = 2 : \langle\mathbb{R}, 2 \approx id_{\mathbb{R}}\rangle \to \langle\mathbb{R}, \mathsf{true}_{\mathbb{R}}\rangle$, it is the case that $s^\sharp A = t^\sharp A = \langle p, id_{\mathbb{R}}\rangle_{\asymp} \in \mathcal{P}(\langle\mathbb{R}, 2 \approx id_{\mathbb{R}}\rangle)$. Hence, $\mathcal{P}_{\mathbb{J},\mathcal{D}}$ cannot be handled by previous frameworks of the same authors [24, 2], which only deal with free-goal programs.

We want to analyze the derivation steps which are available in our framework. In particular, we want to check whether categorical derivations correspond to SLD derivations in standard CLP languages. Hence we will restrict our attention to atomic programs. It is easy to check that these are the only possible derivation steps:

- using a clause reduction pair, we have

$$\mathbf{G}_1, p(t), \mathbf{G}_2 \xrightarrow{\langle r, \langle \mathbf{G}_1, s, cl, \mathbf{G}_2\rangle\rangle} r^\sharp\mathbf{G}_1, s^\sharp\mathbf{Tl}, r^\sharp\mathbf{G}_2 \ , \tag{7.5}$$

  when $cl : p(t') \leftarrow \mathbf{Tl}$ of sort $\langle\sigma', c'\rangle$, $r : \langle\rho, d\rangle \to \langle\sigma, c\rangle$, $s : \langle\rho, d\rangle \to \langle\sigma', c'\rangle$ and $\langle r, s\rangle$ is a unifier of $p(t)$ and $p(t')$;

- using an arrow reduction pair, we have

$$\mathbf{G} \xrightarrow{\langle r, id_{r^\sharp\mathbf{G}}\rangle} r^\sharp\mathbf{G} \ , \tag{7.6}$$

  where $r : \langle\rho, d\rangle \to \langle\sigma, c\rangle$.

The latter does not correspond to any step in standard CLP languages: it is an arbitrary instantiation of the goal with the simultaneous addition of c' to the constraint store. If we consider only maximal reduction pairs, then $r = id$ and the step is harmless since it does not change the current goal and has an identity answer.

We want to give more attention to the case of clause reduction pairs, by analyzing the structure of the unifiers of $p(t_1) : \langle \sigma_1, c_1 \rangle$ and $p(t_2) : \langle \sigma_2, c_2 \rangle$. Assume there is a span $\langle r_1 : \alpha \to \sigma_1, r_2 : \alpha \to \sigma_2 \rangle$ in $\mathbb{C}$. If this span is a unifier of $p(t_1)$ and $p(t_2)$ in $\mathcal{P}_{\mathbb{J}}$, i.e. $r_1 t_1 = r_2 t_2$, then it yields an obvious unifier in $\mathcal{P}_{\mathbb{J},\mathcal{D}}$, namely $\langle r_1, r_2 \rangle$ where $r_1 : \langle \alpha, {r_1}^\sharp c_1 \wedge {r_2}^\sharp c_2 \rangle \to \langle \sigma_1, c_1 \rangle$ and $r_2 : \langle \alpha, {r_1}^\sharp c_1 \wedge {r_2}^\sharp c_2 \rangle \to \langle \sigma_2, c_2 \rangle$. However, also when $r_1 t_1 \neq r_2 t_2$ in $\mathbb{C}$, $\langle r_1, r_2 \rangle$ is still a unifier in $\mathcal{P}_{\mathbb{J},\mathcal{D}}$, if $r_1 : \langle \alpha, {r_1}^\sharp c_1 \wedge {r_2}^\sharp c_2 \wedge (r_1 t_1 \approx r_2 t_2) \rangle \to \langle \sigma_1, c_1 \rangle$ and $r_2 : \langle \alpha, {r_1}^\sharp c_1 \wedge {r_2}^\sharp c_2 \wedge (r_1 t_1 \approx r_2 t_2) \rangle \to \langle \sigma_2, c_2 \rangle$. More generally:

Proposition 7.7 *A unifier of $p(t_1) : \langle \sigma_1, c_1 \rangle$ and $p(t_2) : \langle \sigma_2, c_2 \rangle$ is a span $\langle r_1, r_2 \rangle$ with domain $\langle \alpha, d \rangle$ such that $d \leq {r_1}^\sharp c_1 \wedge {r_2}^\sharp c_2 \wedge r_1 t_1 \approx r_2 t_2$.*

For example, let us consider $\mathcal{P}_{\mathbb{J},\mathcal{R}}$, and assume given goals $p(id) : \langle \mathbb{R}, \mathsf{true} \rangle$ and $p(\lambda x.3) : \langle \mathbb{R}, \mathsf{true} \rangle$. Among the many unifiers, there are the following pairs:

$$r_1 : \langle \mathbb{R}, \mathsf{true} \rangle \xrightarrow{\lambda x.3} \langle \mathbb{R}, \mathsf{true} \rangle \;\; r_2 : \langle \mathbb{R}, \mathsf{true} \rangle \xrightarrow{id} \langle \mathbb{R}, \mathsf{true} \rangle \;\; ,$$

$$r'_1 : \langle \mathbb{R}, id \approx \lambda x.3 \rangle \xrightarrow{id} \langle \mathbb{R}, \mathsf{true} \rangle \;\; r'_2 : \langle \mathbb{R}, id \approx \lambda x.3 \rangle \xrightarrow{id} \langle \mathbb{R}, \mathsf{true} \rangle \;\; .$$

The first is a unifier since ${r_1}^\sharp p(id) = p(\lambda x.3) = {r_2}^\sharp p(\lambda x.3)$ in $\mathcal{P}_{\mathbb{J}}$. The second is a unifier since, although ${r'_1}^\sharp p(id) = p(id)$ and ${r'_2}^\sharp p(\lambda x.3) = p(\lambda x.3)$ are different in $\mathcal{P}_{\mathbb{J}}$, in the fiber corresponding to the domain of r'_1 the constraint $id \approx \lambda x.3$ is true, hence the two goals are in the same equivalence class.

In particular, if $\mathcal{D}$ has inconsistent states, then $\mathsf{false} \leq t_1 \approx t_2$ for each equational constraint $t_1 \approx t_2$. Therefore, if we want to unify $p(t_1)$ and $p(t_2)$, it is enough to move to the fiber $\langle \sigma_1 \times \sigma_2, \mathsf{false} \rangle$, since there $p(t_1)$ and $p(t_2)$ are in the same equivalence class. For example, with the clause $p(3) \leftarrow \top$ of type $\langle \mathbf{1}, \mathsf{true} \rangle$, the goal $p(2) : \langle \mathbf{1}, \mathsf{true} \rangle$ succeeds with answer $id : \langle \mathbf{1}, \mathsf{false} \rangle \to \langle \mathbf{1}, \mathsf{true} \rangle$. Although this behavior is logically faithful, we could dislike it from an operational point of view. In this case, we should change the definition of $\mathbb{D}$, according to the Section 7.1.1.

The good point is that unifiers in $\mathcal{P}_{\mathbb{J},\mathcal{D}}$ have terminal elements, also when unifiers in $\mathcal{P}_{\mathbb{J}}$ do not. We have the following:

Proposition 7.8 *Given $p(t_1) : \langle \sigma_1, c_1 \rangle$ and $p(t_2) : \langle \sigma_2, c_2 \rangle$, if π_1 and π_2 denote the projections in $\mathbb{C}$ from $\sigma_1 \times \sigma_2$ to σ_1 and σ_2 respectively, then the span $\langle \pi_1, \pi_2 \rangle$ with domain $\langle \sigma_1 \times \sigma_2, {\pi_1}^\sharp c_1 \wedge {\pi_2}^\sharp c_2 \wedge \pi_1 t_1 \approx \pi_2 t_2 \rangle$ is a terminal unifier.*

Proof. The span $\langle \pi_1, \pi_2 \rangle$ is a unifier thanks to Proposition 7.7. We need to prove it is terminal. Assume $\langle r_1, r_2 \rangle$ with domain $\langle \alpha, d \rangle$ is another unifier of $p(t_1)$ and $p(t_2)$. In $\mathbb{C}$, there is a unique $h : \alpha \to \sigma_1 \times \sigma_2$ such that $h\pi_1 = r_1$ and $h\pi_2 = r_2$ which is given by the universal property of $\sigma_1 \times \sigma_2$. Therefore,

$$\frac{\Gamma \vdash_{\mathcal{C}} c}{\Delta; \Gamma \vdash c}\ (C_R) \qquad \frac{\Gamma \vdash_{\mathcal{C}} A \sim A'}{\Delta, A; \Gamma \vdash A'}\ (Atom)$$

$$\frac{\Delta; \Gamma \vdash G_1 \qquad \Delta; \Gamma \vdash G_2}{\Delta; \Gamma \vdash G_1 \wedge G_2}\ (\wedge_R) \qquad \frac{\Delta; \Gamma \vdash G_1 \qquad \Delta, A; \Gamma \vdash G}{\Delta, G_1 \Rightarrow A; \Gamma \vdash G}\ (\Rightarrow_L)$$

$$\frac{\Delta; \Gamma, c \vdash G[x/y] \qquad \Gamma \vdash_{\mathcal{C}} \exists_y c}{\Delta; \Gamma \vdash \exists_x G}\ (\exists_R) \qquad \frac{\Delta, D[x/y]; \Gamma, c \vdash G \qquad \Gamma \vdash_{\mathcal{C}} \exists_y c}{\Delta, \forall_x D; \Gamma \vdash G}\ (\forall_L)$$

in both, y does not appear free in the sequent of the conclusion.

Figure 2: Inference rules for $\mathcal{IC}$. Here Δ is a set of clauses, Γ is a set of constraints, G, G_1, G_2 are goals, A, A' are atoms, D is a clause.

we only need to check that h is an arrow from the unifier $\langle r_1, r_2 \rangle$ to $\langle \pi_1, \pi_2 \rangle$, i.e that

$$d \leq h^\sharp({\pi_1}^\sharp c_1 \wedge {\pi_2}^\sharp c_2 \wedge \pi_1 t_1 \approx \pi_2 t_2)\ . \tag{7.7}$$

It is the case that

$$h^\sharp({\pi_1}^\sharp c_1 \wedge {\pi_2}^\sharp c_2 \wedge \pi_1 t_1 \approx \pi_2 t_2) = r_1^\sharp c_1 \wedge r_2^\sharp c_2 \wedge r_1 t_1 \approx r_2 t_2\ . \tag{7.8}$$

Then $d \leq r_1^\sharp c_1 \wedge r_2^\sharp c_2 \wedge r_1 t_1 \approx r_2 t_2$ by Proposition 7.7. ∎

For an atomic program, most general unifiers strictly correspond to most general reducers. If we consider most general derivations, they give an operational semantics which is almost equivalent to standard CLP operational semantics, like the one which appears in [40]. The biggest difference is that we do not distinguish between *active* and *passive* constraints.

The theory in Section 5 automatically gives us also denotational and fixpoint semantics for CLP. Doing a detailed comparison of these semantics with the standard ones known in the literature would be quite time-consuming, and we think we have already shown that our reformulation of CLP is faithful. We only want to point out that, while [40] presents two fixpoint semantics, which correspond to ground answer and computed answer semantics of pure logic programs, the Yoneda semantics in Section 6 corresponds to correct answer semantics. Actually, if $\mathbf{G} : \langle \sigma, c \rangle$, then $Y_P(\mathbf{G})$ is the set of arrows $r : \langle \rho, d \rangle \to \langle \sigma, c \rangle$ such that d and P imply $r^\sharp \mathbf{G}$.

7.3. Logical Interpretation

We have already shown that categorical derivation in our CLP framework captures derivation in standard CLP languages. However, we can make the internal logic more precise. Our formalization is faithful to the $\mathcal{IC}$ sequent calculus for constraints defined in [53], restricted to the first-order Horn fragment, shown in Figure 2.

Since [53] is based on syntactic constructions and not on category theory, we need to do some preliminary work to reconcile its notation with ours. In particular, the constraint systems in [53], due to Saraswat [77] are entailment systems in first order logic with equality, conjunction and existential quantifiers,

and possibly other connectives. The correspondence between syntactically presented constraint systems and the categorical presentation of constraint systems we use is fully developed in [68]. However, in order to make this section more self-contained, we give a sketch of this correspondence without proofs.

Given a first order signature (Σ, Π), a *syntactically presented constraint system* is a pair $\mathcal{C} = (\mathcal{L}_\mathcal{C}, \vdash_\mathcal{C})$ where

- $\mathcal{L}_\mathcal{C}$ is a set of first order formulas including true, all equations of the form $t \sim t'$ and closed by conjunction and existential quantifier;
- $\vdash_\mathcal{C}$ is an *entailment* relation which extends entailment in first order intuitionistic logic.

We may turn $\mathcal{C}$ into a constraint system $\mathcal{C}$ over the base category $\mathbb{S}_\Sigma$ such that

- for each object $n \in \mathbb{S}_\Sigma$, the fiber $\mathcal{C}(n)$ is given as follows:
  - constraints in $\mathcal{C}(n)$ are equivalence classes of formulas whose free variables are among $\{v_1, \ldots, v_n\}$, modulo renaming of bound variables; in the following we use formulas instead of equivalence classes, since renaming of bound variables is a congruence w.r.t. all the operations we are going to define;
  - $\phi \leq \psi$ iff $\phi \vdash_\mathcal{C} \psi$;
  - the meet of ϕ and ψ is the constraint $\phi \wedge \psi$;
  - the upper element is true;
- for each substitution $\theta : n \to m$, the reindexing functor $\mathcal{C}(\theta)$ maps the constraint ϕ to $\phi\theta$, after renaming bound variables in ϕ in order to avoid variable clashes;
- for each substitution $\theta : n \to m = \{v_1/t_1, \ldots, v_m/t_m\}$, the left adjoint to $\mathcal{C}(\theta)$ is $\exists^\mathcal{C}_\theta$ and maps the constraint ϕ to

$$\exists^\mathcal{C}_\theta \phi = \exists w_1, \ldots, w_n . v_1 \sim t_1\eta \wedge \ldots \wedge v_m \sim t_m\eta \wedge \phi\eta \ ,$$

  where $\eta = \{v_1/w_1, \ldots, v_n/w_n\}$ and $\{w_1, \ldots, w_n\}$ are fresh variables different from the v_i's.

If we take a discrete display category $\mathbb{J}$ and related functor $\delta : \mathbb{J} \to \mathbb{S}_\Sigma$, we obtain the LP doctrine $\mathcal{P}_{\mathbb{J},\mathcal{C}}$. We show how to map goals and program in the fibers of $\mathcal{P}_{\mathbb{J},\mathcal{C}}$ to formulas in $\mathcal{IC}$. Remember that an atom $\langle p_i, \theta \rangle_\asymp \in \mathcal{P}_{\mathbb{J},\mathcal{C}}(n)$ is an (equivalence class of) pairs made of a predicate symbol p_i of arity n (i.e. $\delta(p_i) = n$) and a substitution $\theta : m \to n$. We denote with $\overline{\theta}$ the tuple of terms $\langle \theta(v_1), \ldots, \theta(v_n) \rangle$, and with $\overline{\langle p_i, \theta \rangle_\asymp}$ the syntactic atom $p_i(\overline{\theta})$. Note that the mapping for atomic goals is not uniquely defined, since different choices of the representative for $\langle p_i, \theta \rangle_\asymp$ yield different syntactic atoms. However, this ambiguity does not affect the correspondence with $\mathcal{IC}$, formalized below.

The goal $\mathbf{G} = A_1, \ldots, A_n$ is mapped to $\overline{\mathbf{G}} = \overline{A_1} \wedge \ldots \wedge \overline{A_n}$ if $n \geq 1$ and to $\overline{\mathbf{G}} = \mathsf{true}$ if $n = 0$. A program P is mapped to $\overline{P} = \{\vec{\forall}(c \wedge \overline{\mathbf{Tl}}) \Rightarrow \overline{\mathbf{Hd}} \mid$

$\mathbf{Hd} \xleftarrow{cl:\langle n,c\rangle} \mathbf{Tl} \in P\}$, where $\vec{\forall}\phi$ is the universal closure of the formula ϕ. We may finally state the correspondence theorem:

Theorem 7.9 *Given an atomic program P in $\mathcal{P}_{\mathbb{J},\mathbb{C}}$, if there is a categorical derivation*

$$\mathbf{G} : \langle \sigma, c\rangle \overset{d}{\rightsquigarrow}{}^{*} \top_{\rho} : \langle \rho, c'\rangle \ ,$$

and $\theta = \mathsf{answer}(d)$, then there exists a proof in $\mathcal{IC}$ of the sequent

$$\overline{P}; \exists^{\mathbb{C}}_{\theta} c' \vdash \overline{\mathbf{G}} \ .$$

The proof may be found in the Appendix.

8. An Example: Enriching the Display Category

Now we will illustrate the scope of the categorical structure introduced in section Section 5 by enriching the structure of the display category $\mathbb{J}$ so as to build-in relations, functions, data types, modules and exceptions directly into Horn Clause syntax.

We proceed in three steps. First we consider non-discrete display categories $\mathbb{J}$. We show how these display categories may be used to construct syntactic LP doctrines with built-in proofs between goals. Later we define *simple display structures* and then *functorial display structures*, which are used to model logic programs which are sensitive to data types in the base category. It should be understood that this information *affects the way the program clauses and predicates will be represented* in the indexed category of proofs. It will generate new predicates, with new relations between them *and new proofs*, i.e. arrows in the fibers. It has no impact, however, on the structure of the base category $\mathbb{C}$, which is defined in advance.

8.1. Non-discrete Display Categories

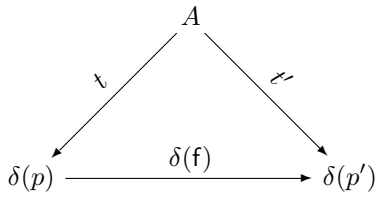
Consider a display category $\mathbb{J}$ and a functor $\delta : \mathbb{J} \to \mathbb{C}$ as in Section 4.1, but without requiring $\mathbb{J}$ to be discrete. We want to extend the definition of $\mathcal{P}_{\mathbb{J}}$ in such a way that arrows in the category $\mathbb{J}$ are mapped to built-in proofs for the goals.

The idea is the following. Assume we have an arrow between predicate symbols $\mathsf{f} : p_1 \to p_2$ in $\mathbb{J}$, and their sorts are σ_1, σ_2 respectively, that is to say $\delta(p_1) = \sigma_1$ and $\delta(p_2) = \sigma_2$. Then, we want this data in $\mathbb{J}$ to force the existence of an arrow $p_1(id_{\sigma_1}) \to p_2(\delta(\mathsf{f}))$ in the fiber $\mathcal{P}(\sigma_1)$. Obviously, this arrow will generate many other arrows in its own fiber and others due to reindexing and the premonoidal structure.

To make sure this kind of display category forces the creation of the right goals and proof-arrows in the resulting indexed category of proofs we need to carry out the construction sketched in Section 4.1, defining an indexed category $(\_ \downarrow \delta)$ by combining two operations, here given a bit more systematically.

Op. 1: Atomic goals and arrows are generated by the comma-category functor. Given two functors $\mathbb{A} \xrightarrow{F} \mathbb{B} \xleftarrow{G} \mathbb{C}$ the comma category $(F \downarrow G)$ is a well-known general device for building new categories from old [57] of which many often used constructions (e.g. the slice) are a special case. We will be interested in the case $(A \downarrow \delta)$ where $\delta : \mathbb{J} \longrightarrow \mathbb{C}$ and where A is an object of $\mathbb{C}$ (which can be viewed as a functor $A : \mathbf{1} \longrightarrow \mathbb{C}$). If A is an object of $\mathbb{C}$, the comma category $(A \downarrow \delta)$ is defined as follows:

- Objects: pairs (p, t) where $p \in |\mathbb{J}|$ and $A \xrightarrow{t} \delta(p)$. These are the same objects we get in the case of discrete display structures.
- Arrows $(p, t) \xrightarrow{f} (p', t')$ are arrows $f : \delta(p) \to \delta(p')$ in $\mathbb{C}$ such that $f = \delta(\mathsf{f})$ for some $\mathsf{f} \in \mathbb{J}$ and the following diagram in $\mathbb{C}$ commutes:



  Arrows compose as in $\mathbb{C}$. This gives us a category of atomic predicates $p(t)$ of sort A represented as arrows $A \xrightarrow{t} \delta(p)$, with new arrows between them imposed by the now non-discrete structure of $\mathbb{J}$.

Now we will allow the first argument of this comma category to range over all objects in $\mathbb{C}$, treating the resulting *variable comma category* $(\_ \downarrow \delta)$ as an indexed category, i.e. a contravariant functor from $\mathbb{C}$ to Cat. We must now define its action on arrows $B \xrightarrow{r} A$ in $\mathbb{C}$, which must be to produce *reindexing* functors $(r \downarrow \delta)$ between the fibers. It acts by composition. $(r \downarrow \delta) : (A \downarrow \delta) \to (B \downarrow \delta)$ maps objects (p, t) to (p, rt), and the arrows $\delta(\mathsf{f})$ that form the base of the commuting triangle above, are sent to themselves, now the base of the commuting triangle formed by $B \xrightarrow{rt} \delta(p)$ and $B \xrightarrow{rt'} \delta(p')$.

Op. 2: Compound goals and arrows are generated by placing premonoidal structure on the fibers. Since the indexed category created by "leaving a parameter blank" in the comma category would only have simple atomic goals, we need to allow sequences of goals and the corresponding new arrows between them. This would be easily achieved by replacing each fiber F with the category F' of finite sequences of objects of F, and arrows by finite sequences of arrows from F, which, in fact, gives the *free monoidal category over* F, with $A_1, \ldots, A_n \otimes B_1, \ldots, B_m = A_1, \ldots, A_n, B_1, \ldots, B_m$. But as the reader may recall from the discussion in Section 3.2 and Section 3.3 such a category would be too liberal, always allowing parallel resolutions $\mathbf{G}_1 \otimes \cdots \otimes \mathbf{G}_n \rightsquigarrow \mathbf{G}'_1 \otimes \cdots \otimes \mathbf{G}'_n$ in which all goals are reduced at once. In order to allow finer control of allowed resolutions we need to build the *free premonoidal* category $\mathbb{A}^*$ generated by a category $\mathbb{A}$. The objects of $\mathbb{A}^*$ are sequences $A_1, \ldots, A_n$ of objects in $\mathbb{A}$, while

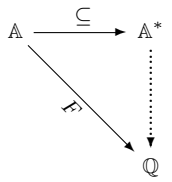
the arrows from $A_1, \ldots, A_n$ to $B_1, \ldots, B_n$ are sequences ξ of pairs (i, f) where $i \in \{1, \ldots, n\}$ and $f \in \mathbb{A}$ is not an identity arrow. Moreover, ξ has to satisfy the following conditions:

1. If $\xi = (i_1, f_1), \ldots, (i_l, f_l)$, for each $j \in \{1, \ldots, n\}$ consider the subsequence $(i_{k_1}, f_{k_1}), \ldots, (i_{k_m}, f_{k_m})$ of ξ made of all the pairs whose first component is j. Then either
   (a) $m = 0$ and $A_j = B_j$, or
   (b) $m \geq 1$, $f_{k_1} \diamond \cdots \diamond f_{k_m}$ does exist and it is an arrow $A_j \to B_j$.
2. There are no consecutive pairs with the same first component.

Arrows compose by juxtaposition. Subsequences such as $(i, f_1)(i, f_2)$,which may originate from juxtaposition, are replaced with the empty sequence when $f_1 f_2 = id$, with $(i, f_1 f_2)$ otherwise. The identity over $A_1, \ldots, A_n$ is the empty sequence.

A premonoidal structure may be defined over $\mathbb{A}^*$. $\top$ is the empty sequence of goals. If $\mathbf{G} = A_1, \ldots, A_n$ and $\xi = (i_1, f_1) \ldots (i_l, f_l)$, then $\mathbf{G} \otimes \xi = (i_1 + n, f_1) \ldots (i_l + n, f_l)$, while $\xi \otimes \mathbf{G} = (i_1, f_1) \ldots (i_l, f_l)$, i.e. the same sequence of pairs, with respect to different domain and codomain.

It really is a free construction: given any functor $F : \mathbb{A} \to \mathbb{Q}$ with $\mathbb{Q}$ premonoidal, there is a unique premonoidal functor from $\mathbb{A}^*$ to $\mathbb{Q}$ making the diagram



commute. The $(\_)^*$ operation may be easily turned into a functor $(\_)^* : \mathsf{Cat} \to \mathsf{Cat}$. If $F : \mathbb{A} \to \mathbb{B}$ is a functor, then $F^* : \mathbb{A}^* \to \mathbb{B}^*$ maps each object $A_1, \ldots, A_n$ to $F(A_1), \ldots, F(A_n)$ and each arrow $(i_1, f_i) \ldots (i_n, f_n)$ to the arrow $(i_1, F(f_1)) \ldots (i_n, F(f_n))$.

Putting it all together. Now, define $\mathcal{P}_{\mathbb{J}} : \mathbb{C}^{\mathrm{o}} \to \mathsf{Cat}$ to be the functor $(\_ \downarrow \delta)^*$ which maps each object A of $\mathbb{C}$ to the free premonoidal category generated by the comma category $(A \downarrow \delta)$. We sum up what has been achieved by reminding the reader of the universal mapping property satisfied by the whole construction. The proof is straightforward.

Theorem 8.1 *Suppose $\mathbb{J}$ is a display category with associated functor $\delta : \mathbb{J} \to \mathbb{C}$, $\mathcal{P}_{\mathbb{J}}$ the $\mathbb{C}$-indexed category it generates, and $\mathcal{Q}$ a $\mathbb{C}$-indexed premonoidal indexed category satisfying the following conditions:*

- *for each $j \in |\mathbb{J}|$, there is a distinguished object $\zeta(\mathsf{j})$ in the fiber $\mathcal{Q}(\delta(\mathsf{j}))$;*
- *for each arrow $j \xrightarrow{\mathsf{a}} k$ in $\mathbb{J}$, there is a distinguished arrow $\zeta(\mathsf{j}) \xrightarrow{m_{\mathsf{a}}} \mathcal{Q}(\delta(\mathsf{a}))(\zeta(k))$.*

Then there is a unique indexed premonoidal functor $\mathcal{P}_{\mathcal{J}} \xrightarrow{\Psi} \mathcal{Q}$ *such that* $\Psi(\mathsf{a}) = m_{\mathsf{a}}$ *for every arrow* $j \xrightarrow{\mathsf{a}} k$ *in* $\mathbb{J}$.

Example 8.2 Consider an FP category $\mathbb{C}$ and a display category $\mathbb{J}$, such that the only non-identity arrow in $\mathbb{J}$ is a map $\mathit{refp} : p \to p$ for some $p \in |\mathbb{J}|$. Let us assume δ maps *refp* to the arrow $\langle \pi_2, \pi_1 \rangle$, where π_1, π_2 are the two projections from $\delta(p) \times \delta(p)$ to $\delta(p)$. Then $\mathcal{P}_{\mathbb{J}}$ is isomorphic to the LP doctrine with symmetric predicate defined in the Example 5.3. ■

8.2. Data Types in the Display Category

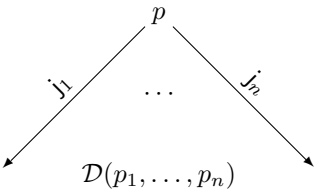
The display category will now be replaced by a structure called the *simple display structure* similar in spirit to the FP and limit *sketches* defined in [9].

Definition 8.3 *A* simple display structure $\mathcal{J} = (\mathbb{J}, \mathcal{C}, \mathcal{K}, \delta)$, *is composed of*

- *a category* $\mathbb{J}$, *a set* $\mathcal{C}$ *of distinguished diagrams from* $\mathbb{J}$, *and a set* $\mathcal{K}$ *of distinguished cones from* $\mathbb{J}$;
- *a functor* $\delta : \mathbb{J} \to \mathbb{C}$ *that must send all diagrams in* $\mathcal{C}$ *to commutative diagrams and all cones in* $\mathcal{K}$ *to cones in* $\mathbb{C}$.

Now we need to modify the definition of $\mathcal{P}_{\mathbb{J}}$ given in Section 4.1 to obtain a new LP doctrine $\mathcal{P}_{\mathcal{J}}$ where cones distinguished in the display structure $\mathcal{J}$ are reflected in a certain way by predicates and proofs. This is accomplished by a process called $\mathcal{K}$-*closure*, defined below.

Definition 8.4 *The* $\mathbb{C}$-*indexed monoidal category generated by a display structure* $\mathcal{J}$ *denoted by* $\mathcal{P}_{\mathcal{J}}$ *is defined to be the premonoidal indexed category obtained from* $\mathcal{P}_{\mathbb{J}}$ *by the following* $\mathcal{K}$-*closure condition in* $\mathcal{P}_{\mathbb{J}}$.
For every cone κ *in* $\mathcal{K}$



over a diagram $\mathcal{D}(p_1, \ldots, p_n)$ *with the object* p_i *of* $\mathcal{D}$ *the target of* j_i, *the following arrow should be freely added to the fiber* $\mathcal{P}_{\mathbb{J}}(\delta(p))$*:*

$$(p_1, \delta(\mathsf{j}_1)) \otimes \cdots \otimes (p_n, \delta(\mathsf{j}_n)) \xrightarrow{\kappa} (p, id_{\delta(p)}) \ . \tag{8.1}$$

Freely adjoining an arrow in a fiber will generate in turn many new proofs, given by formal reindexing along arrows in the base category and by closure w.r.t. the premonoidal structure. We may build $\mathcal{P}_{\mathcal{J}}$ constructively from $\mathcal{P}_{\mathbb{J}}$ in the same way we build the free model $\mathcal{F}_P$ of a program P starting from an LP doctrine $\mathcal{P}$: $\mathcal{P}_{\mathbb{J}}$ plays the role of $\mathcal{P}$, and the set of arrows added by $\mathcal{K}$-closure plays the role of the program P.

A special case of interest is when $\kappa \in \mathcal{K}$ is a single isolated object p, the cone over the empty diagram. Then, since there can be no arrows into the empty diagram, the premise of $\mathcal{K}$-closure is vacuously satisfied. Thus $\top_\alpha \longrightarrow (p, id_\alpha)$ is always in $\mathcal{P}_\mathcal{J}(\alpha)$.

Another special case of interest is when δ maps all cones in $\mathcal{K}$ to limiting cones. If p is the vertex of a cone over a diagram $\mathcal{D}$ in $\mathcal{K}$, the cone is mapped by δ to a cone $\delta(\mathcal{D})$ in $\mathbb{C}$, and $\delta(p)$ is its limit. Thus given any other cone over $\delta(\mathcal{D})$ with vertex α, and arrows $t_1, \ldots, t_n$ from α into $\mathcal{D}$ there is a unique arrow in $\mathbb{C}$ from α to $\delta(p)$ making all diagrams commute. This arrow will be called $lim_\mathcal{D}(t_1, \ldots, t_n)$.

In this case, the effect of the closure condition is that, for every object $\alpha \in \mathbb{C}$ and every cone $\{\alpha \xrightarrow{t_i} \delta(p_i) : 1 \leq i \leq n\}$ over $\delta(\mathcal{D}(p_1, \ldots, p_n))$ in $\mathcal{P}_\mathcal{J}(\alpha)$, there is an arrow

$$(p_1, t_1) \otimes \cdots \otimes (p_n, t_n) \xrightarrow{lim_\mathcal{D}(t_1,\ldots,t_n)^\sharp \kappa} (p, lim_\mathcal{D}(t_1, \ldots, t_n)) \ ,$$

as can be seen by reindexing the arrow (8.1) in $\mathcal{P}_\mathcal{J}(\delta(p))$ along $\alpha \xrightarrow{lim_\mathcal{D}(t_1,\ldots,t_n)} \delta(p)$. If the cone κ is a binary span

$$1 \xleftarrow{\mathsf{l}} 3 \xrightarrow{\mathsf{r}} 2 \ ,$$

then $\delta(\kappa)$ is a product diagram in $\mathbb{C}$, and the $\mathcal{K}$-closure condition just reads as follows: If $\alpha \xrightarrow{t_1} \delta(1)$ and $\alpha \xrightarrow{t_2} \delta(2)$ are arrows in $\mathbb{C}$, then the arrow

$$(1, t_1) \otimes (2, t_2) \longrightarrow (3, \langle t_1, t_2 \rangle)$$

must exist in $\mathcal{P}_\mathcal{J}(\alpha)$, where $\langle t_1, t_2 \rangle$ is the canonical product arrow in $\mathbb{C}$ from α to $\delta(3) = \delta(1) \times \delta(2)$.

We sum up what has been achieved by reminding the reader of the universal mapping property satisfied by the whole construction. The proof is straightforward.

Theorem 8.5 *Suppose $\mathcal{J}$ is a simple display structure, $\mathcal{P}_\mathcal{J}$ the $\mathbb{C}$-indexed category it generates, and $\mathcal{Q}$ a $\mathbb{C}$-indexed premonoidal indexed category satisfying the following conditions:*

- *for each $j \in |\mathbb{J}|$, there is a distinguished object $\zeta(\mathsf{j})$ in the fiber $\mathcal{Q}(\delta(\mathsf{j}))$;*
- *for each arrow $j \xrightarrow{\mathsf{a}} k$ in $\mathbb{J}$, there is a distinguished arrow $\zeta(\mathsf{j}) \xrightarrow{m_\mathsf{a}} \mathcal{Q}(\delta(\mathsf{a}))(\zeta(k))$;*
- *($\mathcal{K}$-closure) for each cone κ over a diagram $\mathcal{D}$ in $\mathcal{K}$ with vertex p and arrows $\{p \xrightarrow{\mathsf{j}_i} p_i \mid 1 \leq i \leq n\}$, there is a distinguished arrow*

$$\mathcal{Q}(\delta(\mathsf{j}_1))(\zeta(p_1)) \otimes \cdots \otimes \mathcal{Q}(\delta(\mathsf{j}_n))(\zeta(p_n)) \xrightarrow{m_\kappa} \zeta(p) \ .$$

Then there is a unique indexed premonoidal functor $\mathcal{P}_{\mathcal{J}} \xrightarrow{\Psi} \mathcal{Q}$ *such that* $\Psi(\mathsf{a}) = m_{\mathsf{a}}$ *for every arrow* $j \xrightarrow{\mathsf{a}} k$ *in* $\mathbb{J}$*, and* $\Psi(\kappa) = m_\kappa$ *for every arrow* κ *induced by* κ*-closure.*

Before describing how to use display structures to add polynomial data types to an indexed logic programming category, we work through an example.

Example 8.6 (Integer Lists) This example illustrates how a suitable choice of display structure will give rise to atomic predicates that respect the `nat list` data type

```
datatype nat list = nil | cons of nat * nat list
```

in the following sense: corresponding to a chosen predicate symbol p of type `nat` occurring in the program, a new predicate symbol $list(p)$ of type `nat list` will be created with the following properties.

- $list(p)(nil)$ has a successful derivation.
- Whenever $p(t)$ and $list(p)(l)$ have a successful derivation, so does the goal $list(p)(cons(t, l))$.

Assume that the base category of types $\mathbb{C}$ has all finite products, a terminal object, the natural numbers $\mathbb{N}$ and an object $\mathbb{N}^*$ (isomorphic to) the set of lists of natural numbers, together with the arrows

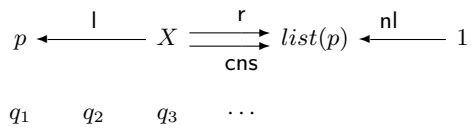
$$cons : \mathbb{N} \times \mathbb{N}^* \longrightarrow \mathbb{N}^* \qquad nil : \mathbf{1} \longrightarrow \mathbb{N}^*$$

as usually defined. $\mathbb{C}$ might also contain enough arrows into and out of $\mathbb{N}$ and $\mathbb{N}^*$ to ensure availability of needed operations, and arrows to interpret all function symbols in the signature of the programs of interest, e.g. all $\mathbf{1} \xrightarrow{n} \mathbb{N}$ for natural numbers n, and all ground integer lists $\mathbf{1} \longrightarrow \mathbb{N}^*$, and possibly $+$ and $\times : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$.

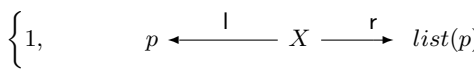
In addition to the predicate symbol p on natural numbers, we assume the program or programs of interest contain predicate symbols $q_1, q_2, \ldots$ of types $\sigma_1, \sigma_2, \ldots$, which are objects of $\mathbb{C}$. All uninterpreted or built-in function symbols and constant symbols occurring in the program, as always, are assumed to be represented by arrows in $\mathbb{C}$.

Let the display structure $\mathcal{J} = (\mathbb{J}, \mathcal{C}, \mathcal{K}, \delta)$ be as follows:

- $\mathbb{J}$ is the category shown (with identity arrows omitted):



- $\mathcal{C} = \emptyset$.
- The set of distinguished cones $\mathcal{K}$ is



Note that the object 1 is the cone over the empty diagram.

- The functor $\delta : \mathbb{J} \to \mathbb{C}$ is defined as follows:

$$\delta(X \xrightarrow{\mathsf{l}} p) = \mathbb{N} \times \mathbb{N}^* \xrightarrow{\pi_1} \mathbb{N} \qquad \delta(X \xrightarrow{\mathsf{r}} list(p)) = \mathbb{N} \times \mathbb{N}^* \xrightarrow{\pi_2} \mathbb{N}^*$$
$$\delta(X \xrightarrow{\mathsf{cns}} list(p)) = \mathbb{N} \times \mathbb{N}^* \xrightarrow{cons} \mathbb{N}^* \qquad \delta(1 \xrightarrow{\mathsf{nl}} list(p)) = \mathbf{1} \xrightarrow{nil} \mathbb{N}^*$$
$$\delta(q_1) = \sigma_1 \qquad \delta(q_2) = \sigma_2 \qquad \ldots$$

  Thus δ sends the diagrams in $\mathcal{K}$ to the respective limiting cones $\mathbf{1}$ (the terminal object of $\mathbb{C}$) and the product diagram for $\mathbb{N} \times \mathbb{N}^*$.

The proofs generated by $\mathcal{J}$. The indexed premonoidal category of goals $\mathcal{P} = \mathcal{P}_{\mathcal{J},P}$ will include:

- In the fiber over σ_i, the object $q_i(id_{\sigma_i})$ representing the predicate symbol q_i. For any arrow $\rho \xrightarrow{t} \sigma_i$, the instance $q_i(t)$ in the fiber $\mathcal{P}(\rho)$. In the fiber $\mathcal{P}(\mathbb{N})$ the object $p(id_{\mathbb{N}})$.
- In the fiber $\mathcal{P}(\delta(list(p))) = \mathcal{P}(\mathbb{N}^*)$ we have an object $list(p)(id_{\mathbb{N}^*})$ representing the predicate $list(p)$.
- In the fiber $\mathcal{P}(\delta(X)) = \mathcal{P}(\mathbb{N} \times \mathbb{N}^*)$ we have objects $X(id_{\mathbb{N}\times\mathbb{N}^*})$ representing a new predicate we will call $p \cdot list(p)$. Since $\pi_1 : \mathbb{N} \times \mathbb{N}^* \to \mathbb{N}$ and $\delta(p) = \mathbb{N}$, in the fiber $\mathcal{P}(\mathbb{N} \times \mathbb{N}^*)$ we also find $p(\pi_1)$, the reindexed predicate p now treated as a predicate of type $\mathbb{N} \times \mathbb{N}^*$ with a hidden variable of type $\mathbb{N}^*$. Finally, for similar reasons we find $list(p)(\pi_2)$ and $list(p)(cons)$.
- In the fiber $\mathcal{P}(\delta(1)) = \mathcal{P}(\mathbf{1})$ the predicate $list(p)(nil)$.
- In every fiber $\mathcal{P}(\sigma)$ the arrows resulting from $\mathcal{K}$-closure. In particular, there is an arrow $\top \longrightarrow (1, !_\sigma)$ because of the presence of a cone mapped to the terminal object of $\mathbb{C}$, and, if $\sigma \xrightarrow{t} \delta(1)$ and $\sigma \xrightarrow{l} \delta(list(p))$ are arrows in $\mathbb{C}$, then the arrow
$$p(t) \otimes list(p)(l) \longrightarrow p \cdot list(p)(\langle t, l\rangle)$$
  exists in $\mathcal{P}(\sigma)$. This means that if $p(t)$ and $list(p)(l)$ have successful derivations, then so does $p(t) \otimes list(p)(l)$ by the premonoidal structures of derivations, hence $p \cdot list(p)(\langle t, l\rangle)$ has a successful derivation too.
- Finally, the presence of the arrows $X \xrightarrow{\mathsf{cns}} list(p)$ and $1 \xrightarrow{\mathsf{nl}} list(p)$ (with $\delta(\mathsf{cns}) = cons$ and $\delta(\mathsf{nl}) = nil$) in $\mathcal{J}$ and the functoriality of $(\_ \downarrow \delta)$ force the existence of the arrows $p \cdot list(p)(\langle t, l\rangle) \xrightarrow{cons} list(p)(cons\langle t, l\rangle)$ and $\top \longrightarrow list(p)(nil)$. If $p(t)$ and $list(p)(l)$ have successful derivations, then
$$list(p)(cons\langle t, l\rangle) \xrightarrow{cons\rightsquigarrow} p \cdot list(p)(\langle t, l\rangle) \rightsquigarrow p(t) \otimes list(p)(l) \rightsquigarrow^* \top \ .$$
  Thus $list(p)(nil)$ and $list(p)(cons\langle t, l\rangle)$ have successful derivations.

■

Example 8.7 We can add a function t to the list data type with a slight modification of the display category and automatically generate the appropriate new proofs in the fibers incorporating it into resolution. We do so in the spirit of logic programming rather than functional programming, by generating a functional *predicate* `pred[t]` corresponding to the introduced function. In this way, we are able to encapsulate data types and capture modules in display structures. We illustrate with an example. Consider the module

```
begin module natlist
  datatype nat list = nil | cons of nat * nat list
  fun length nil = 0
   |  length (cons(a,x)) = 1 + length x;
end module
```

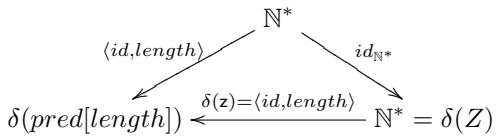
To the display structure of the preceding example we now add

- to $\mathbb{J}$ a couple of new objects Z and $pred[length]$, as well as a new arrow $Z \xrightarrow{\mathsf{z}} pred[length]$; we also define

$$\delta(Z \xrightarrow{\mathsf{z}} pred[length]) = \mathbb{N}^* \xrightarrow{\langle id,length\rangle} \mathbb{N}^* \times \mathbb{N} \ ;$$

- to $\mathcal{K}$ the cone (over the empty diagram) given by single object Z.

We will have the following commutative diagram in $\mathbb{C}$, where *length* is assumed to be defined as an arrow $\mathbb{N}^* \to \mathbb{N}$ satisfying the equations in the definition of the module given above:



which yields a new arrow $Z \xrightarrow{z} pred[length](\langle id, length\rangle)$ in the fiber $\mathcal{P}(\mathbb{N}^*)$.

Note that δ does *not* send the cone Z over the empty diagram to a limit cone (otherwise its image would have to be $\mathbf{1}$) but the $\mathcal{K}$ closure condition adds to the fiber over $\mathbb{N}^*$ the arrow

$$\top_{\mathbb{N}^*} \longrightarrow Z.$$

Hence we have an arrow $\top_{\mathbb{N}^*} \longrightarrow Z \xrightarrow{z} pred[length](id, length)$, which corresponds to a derivation

$$pred[length](X, length(X)) \rightsquigarrow^* \top_{\mathbb{N}} \ .$$

■

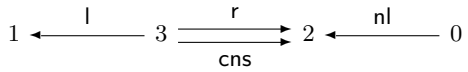
The examples were chosen because of their simplicity, but with some additional display parameters, the method can be applied to all polynomial polymorphic data types, and a variety of interesting monads and module definitions that are not easily coded into a first-order language, with, of course, the added generality of a categorical notion of unification.

In Section 8.3 we will show briefly how to add generic data types, such as `list`, to $\mathcal{P}$. The idea is that we do not want to add just the type `nat list`, but all the types of the form X `list` for any data type X. This means also iterated applications of the list constructor, such as `nat list list`. Moreover, the predicate $list(p)$ should be defined for each predicate p in the display structure.

8.3. Functorial Display Structures

We first illustrate with the polymorphic list type what we will then do with the generic definition of a polymorphic polynomial data type T. We will only consider the case of polymorphic data types that are definable via a collection of product diagrams in the display category, although related techniques have been shown to work for arbitrary monads in [60].

The idea is to use a new display structure which is not mapped to objects and arrows in $\mathbb{C}$, but to *functors* and *natural transformations* in $\mathbb{C}^{\mathbb{C}}$. In the case of the polymorphic list type, the new display structure would be the tuple $(\mathbb{F}, \emptyset, \mathcal{F}, \eta)$ where $\mathbb{F}$ is the following category (with identity arrows omitted)



with distinguished cones 0 and $1 \xleftarrow{\pi_1} 3 \xrightarrow{\pi_2} 2$. The functor $\eta : \mathbb{F} \to \mathbb{C}^{\mathbb{C}}$ is defined as follows:

$$\begin{array}{lcllcl} \eta(3 \xrightarrow{\mathsf{hd}} 1) & = & id \times list \xrightarrow{\pi_1} id & \eta(3 \xrightarrow{\mathsf{r}} 2) & = & id \times list \xrightarrow{\pi_2} list \\ \eta(3 \xrightarrow{\mathsf{cns}} 2) & = & id \times list \xrightarrow{cons} list & \eta(0 \xrightarrow{\mathsf{nl}} 2) & = & \mathbf{1} \xrightarrow{nil} list \ , \end{array}$$

where $id : \mathbb{C} \to \mathbb{C}$ is the identity functor, $list : \mathbb{C} \to \mathbb{C}$ is the functor that, given a type σ, returns the type $list(\sigma)$ and which sends arrow $f : \sigma \to \rho$ to the map $list(f)$ which is the pointwise extension of f to lists.

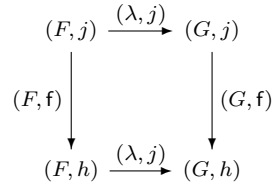
This new kind of display structure may be coupled with a simple display structure which gives names to predicates and optionally defines other non-polymorphic data types. The complete structure obtained so far is called *functorial display structure*. For technical reasons, we only consider functorial display structures over a small category, so that all our definitions are within the realm of locally small categories.

Definition 8.8 *A* functorial display structure $\mathcal{G} = (\mathbb{J}, \mathcal{C}, \mathcal{K}, \delta, \mathbb{F}, \mathcal{F}, \eta)$ *over a small category* $\mathbb{C}$ *is a pair of display structures, the first,* $(\mathbb{J}, \mathcal{C}, \mathcal{K}, \delta)$*, over* $\mathbb{C}$*, and the second,* $(\mathbb{F}, \emptyset, \mathcal{F}, \eta)$*, over* $\mathbb{C}^{\mathbb{C}}$*, that is to say,* η *is a functor from* $\mathbb{F}$ *to the category of endofunctors and natural transformations on* $\mathbb{C}$*. This display structure is assumed to have* only n-ary spans *in its set of cones* $\mathcal{F}$ *and an empty set of distinguished diagrams.*

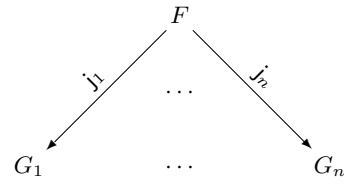
Given a functorial display structure $\mathcal{G}$ we define the LP doctrine $\mathcal{P}_{\mathcal{G}}$ it induces by constructing a simple display structure having the same effect. This definition can be extended to the case where the second display structure has a distinguished set of commutative diagrams and arbitrary functorial cones. Also, the resulting indexed category of proofs can be defined by working directly with the functor category $\mathbb{C}^{\mathbb{C}}$, but the techniques are beyond the scope of the paper.

Definition 8.9 $\mathcal{J}(\mathcal{G}) = (\mathbb{H}, \mathcal{C}_H, \mathcal{K}_H, \hat{\delta})$*, the simple display structure associated with the functorial display structure* $\mathcal{G}$ *is defined as follows.*

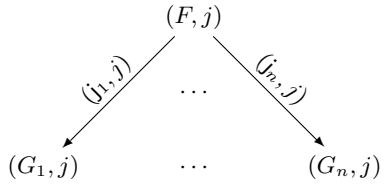
1. *The category* $\mathbb{H}$ *consists of the smallest class of arrows containing those of* $\mathbb{J}$ *and closed under the following conditions.*
   (a) *For each arrow (i.e. natural transformation)* $F \xrightarrow{\lambda} G$ *in* $\mathbb{F}$ *and each* $j \in |\mathbb{H}|$ *there is an arrow* $(F, j) \xrightarrow{(\lambda, j)} (G, j)$ *in* $\mathbb{H}$. *Composition is given by* $(\lambda, j)(\nu, j) = (\lambda\nu, j)$.
   (b) *For each* $j \xrightarrow{\mathsf{f}} h$ *in* $\mathbb{H}$ *and* $F \in |\mathbb{F}|$ *there is an arrow* $(F, j) \xrightarrow{(F,\mathsf{f})} (F, h)$ *in* $\mathbb{H}$. $(F, \mathsf{f}_1)(F, \mathsf{f}_2) = (F, \mathsf{f}_1\mathsf{f}_2)$.
2. $\mathcal{C}_H$ *is the least set of diagrams containing* $\mathcal{C}$ *and closed under the following condition. For each* $j \xrightarrow{\mathsf{f}} h$ *in* $\mathbb{H}$ *and* $F \xrightarrow{\lambda} G$ *in* $\mathbb{F}$ *the following naturality diagram is in* $\mathcal{C}_H$.



3. $\mathcal{K}_H$ *is the least set containing* $\mathcal{K}$ *and closed under the following condition. For each span in* $\mathcal{F}$



and every $j \in |\mathbb{H}|$ *the following diagram*



is in $\mathcal{K}_H$.
4. *The functor* $\hat{\delta}$ *acts as follows:* $\hat{\delta}|_{\mathbb{J}} = \delta$, $\hat{\delta}(F, j) = \eta(F)(\hat{\delta}(j))$, *and* $\hat{\delta}(\lambda, j) = \eta(\lambda)_{\hat{\delta}(j)}$.

8.3.1. Polynomial Data Types

We will now show how to construct the functorial display structure $\mathcal{G}$ that will guarantee, for a polymorphic polynomial data type T, such as the polymorphic `list` type, the existence, in $\mathcal{P}_{\mathcal{J}(\mathcal{G})}$, of predicates $T(p)$, $T(T(p))$, etc. for all predicate symbols p of any type, and the corresponding "built-in" proofs as illustrated for predicates p and $list(p)$ of type `nat` and `nat list` above.

Assume that T is given by the following grammar, in sml-style syntax

```
datatype 'a T   =   k0 | k1 ··· | t1 of E1  |··· | tn of En   (8.2)
```

where the `Ei` are products of type terms of the form `R1 * ⋯ * Rm`, all of whose factors are terms built up using T or predefined data type functors over `'a`, basic types (built-in or predefined types) or `'a T`.

In the category $\mathbb{C}$, (8.2) corresponds to a functor $T : \mathbb{C} \to \mathbb{C}$ and a natural iso $\eta : 1 + \ldots + 1 + E_1 + \ldots + E_n \to T$, where 1 is the constant functor, and the E_i's denote functors which are products $E_i^1 \times \ldots \times E_i^{n_i}$ built from the endofunctors id, T or constant functors. By properties of coproducts, $\eta = [k_1, \ldots, k_n, t_1, \ldots, t_n]$ where $k_i : 1 \to T$ and $t_i : E_i \to T$. The functor T is the abstract data type corresponding to the definition (8.2) (see [10] for full details). $\mathtt{T}, \mathtt{1}, \mathtt{E_i}, \mathtt{E_i^j}, \mathtt{k}_i$ and $\mathtt{t}_i$ will be objects and arrow in the display category, mapped by η to the corresponding entities in $\mathbb{C}$.

Example 8.10 The polymorphic *tree* data type

```
datatype 'a tree = leaf of 'a | node of 'a * 'a tree * 'a tree
```

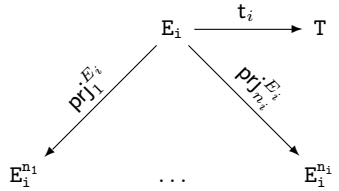
has no constant symbols. The expressions E_i in this case are: $E_1 = id$ and $E_2 = id \times tree \times tree$. ■

We will suppose that a program P has been given, with predicates $p_1, \ldots, p_n$ of types $\sigma_1, \ldots, \sigma_n$, respectively. The functorial display structure for T is $\mathcal{G}_T = (\mathbb{J}, \mathcal{C}, \mathcal{K}, \delta, \mathbb{F}, \mathcal{F}, \eta)$ where $\mathbb{J}$ is the discrete category containing the program predicates $p_1, \ldots, p_n$, mapped to their types in $\mathbb{C}$ by δ, with $\mathcal{C}$ and $\mathcal{K}$ empty and where

- $\mathbb{F}$ must contain the arrows

  $$1 \xrightarrow{\mathtt{k}_0} \mathtt{T} \qquad 1 \xrightarrow{\mathtt{k}_1} \mathtt{T} \quad \cdots \quad 1 \xrightarrow{\mathtt{k}_m} \mathtt{T}$$

  and for each component $\mathtt{E_i}$ of the data type definition for $\mathtt{T}$

  where the same object `T` is common to all the diagrams, and the prj's are arrows which will be mapped to projections (see the $\mathcal{F}$ component of the functorial display structure).

- $\mathcal{F}$ contains the n_i-ary spans obtained by removing the arrows $\mathtt{E_i} \xrightarrow{\mathtt{t}_i} \mathtt{T}$ from the preceding diagram.

- η maps each of $\mathtt{T}, \mathtt{1}, \mathtt{E_i}, \mathtt{E_i^j}, \mathtt{k}_i$ and $\mathtt{t}_i$ to the corresponding functor or natural transformation in $\mathbb{C}$.

As with the list example, there will now be, for every predicate symbol p_i in the program, not only the object (p_i, id_{σ_i}) in the fiber $\mathcal{P}(\sigma_i)$ of the indexed premonoidal category induced by $\mathcal{G}_T$, but also objects $((\mathtt{T}, p_i), id_{T(\sigma_i)})$ representing the induced predicate symbol $\mathtt{T}(p_i)$, $((\mathtt{T}, (\mathtt{T}, p_i)), id_{T(T(\sigma_i))})$, representing $\mathtt{T}(\mathtt{T}(p_i))$, etc. As with $p \cdot list(p)$ above, the products $\mathtt{E_i}$ will induce the corresponding product predicates, which, to avoid cumbersome notation, we will call $\mathtt{E_i}(p)$.

For every induced predicate symbol of the form $\mathtt{T}(Q)$, where Q is either a program predicate symbol p_i or iterates of $\mathtt{T}$ applied to one, we will have successful derivations of the form

$$\mathtt{T}(Q)(\langle u_1, \ldots, u_{n_j}\rangle \diamond \hat{\delta}(\mathtt{t}_j, Q)) \xrightarrow{\langle \hat{\delta}(\mathtt{t}_j, Q), \iota\rangle} \rightsquigarrow \mathtt{E_j}(Q)(\langle u_1, \ldots, u_{n_j}\rangle) \xrightarrow{\langle id, \kappa\rangle} \rightsquigarrow$$
$$\mathtt{E}^1_\mathtt{j}(Q)(u_1) \otimes \cdots \otimes \mathtt{E}^{\mathtt{n_j}}_\mathtt{j}(Q)(u_{n_j}) \rightsquigarrow^* \top$$

whenever all of $\mathtt{E}^1_\mathtt{j}(Q)(u_1), \ldots, \mathtt{E}^{\mathtt{n_j}}_\mathtt{j}(Q)(u_{n_j})$ have successful derivations. In the derivation, ι is the arrow from $\mathtt{E_j}(Q)(\langle u_1, \ldots, u_{n_j}\rangle)$ to $\mathtt{T}(Q)(\langle u_1, \ldots, u_{n_j}\rangle \diamond \hat{\delta}(\mathtt{t}_j, Q))$ introduced in $\mathcal{P}_{\mathcal{J}(\mathcal{G})}$ by the existence of the arrow $(\mathtt{t_i}, \mathtt{Q})$ in $\mathcal{J}(\mathcal{G})$, while κ is created by $\mathcal{K}$ closure.

Example 8.11 (Exceptions) We illustrate how these techniques can be used to formalize exceptions in logic programming. We consider the definition

```
monad 'a exception = exc of int*string | safe of 'a        (8.3)
```

We will omit discussion of how the monadic structure (map, unit, join) associated with this data type can be enforced in a functional display structure $\mathcal{G}_{exc}$. The interested reader should consult [60] where techniques for enforcing monad laws in the display structure and the associated syntactic category are studied in a related setting.

Just by formalizing the data type definition as done above in a functorial display structure $\mathcal{G}$ we obtain that for each program predicate symbol p of type σ, any integer n and string s, $exception(p)(exc_\sigma(n, s))$ is derivable, and if $p(t)$ is derivable, then so is $exception(p)(safe_\sigma(t))$. ■

9. Related Work

This paper builds upon ideas developed in [23, 55, 2, 24, 16, 17]. We integrate and generalize these different proposals, in order to obtain a new algebraic framework which is simpler yet more expressive, in some specific ways we will briefly discuss here.

The foundation for logic programming introduced by Finkelstein, Freyd, Lipton in [23] is a special case of our syntactic LP doctrines $\mathcal{P}_\mathbb{J}$ for a discrete display category $\mathbb{J}$, but with some components of the indexed category structure not made explicit, and with conjunction of goals only modeled with intersections (i.e. idempotent products). As we already explained in Section 3.2, this severely limits the level of operational fidelity it is possible to reach with logic programming practice. It may be considered the principal ancestor of our paper, since it

shares with ours the focus on both operational and fixpoint semantics, although the latter is limited to what is called the Yoneda semantics in our paper.

We find the premonoidal indexed categories approach to categorical logic more appropriate for logic programming analysis, both operationally and semantically. Indexed categories are easier to deal with, since they separate the predicate logic component (in the fibers) from the term, constraint and state components (in the base). This is what allowed us to easily formulate CLP and languages with abstract data types as instances of our framework.

In [24, 47] Finkelstein, Freyd, Lipton and Krishnan extend the categorical approach of [23] to First Order Hereditarily Harrop Formulas, taking advantage of the indexed category structure to model the changing ambient program in the base. Extending the logic itself is outside the scope of the current treatment for reasons discussed in the introduction, but that is probably the next natural step in our research.

Other key precursors of the ideas in this paper, especially as regards display structures, modules and data types are found in [55, 60] which extend [23] by introducing modules and abstract data types. This is what this paper does using simple and functorial display structures, with a far more general treatment of proofs and models, and with premonoidal categories used to model conjunctions. In [60] McGrail also extends [23] with monads, which are automatically lifted from the category of terms to the logic level. These results need to be extended to the premonoidal framework given in this paper.

Along a different line of research, Corradini and Asperti [16] and Kinoshita and Power [45] give categorical semantics for logic programs based on indexed monoidal categories and indexed finite product categories respectively. These papers have a similar approach at the operational level, but do not consider any fixpoint semantics. Moreover, the use of monoidal and finite product structures, instead of premonoidal ones, reduces the operational fidelity to real-world logic programming, since their notions of proof allow parallel resolution instead of standard one-goal-at-a-time resolution. Also, they only consider the special case of generic predicates with a base category of sorts, corresponding to discrete display categories in our settings.

Note that [16] is based on some ideas already developed by Corradini and Montanari [17]. In the latter, the focus was on a generic algebraic methodology to derive structured transition systems and models for a large class of formalisms. Logic programming was the main example. Given a logic program P, the paper shows how to build a structured transition system (corresponding to $\rightsquigarrow$ in our framework) and a class of models for P. The paper does not deal with fixpoint semantics and only considers pure (syntactic) logic programs. Products are used to model conjunctions. Moreover, the paper does not use indexed categories, but a sort of layering between terms and predicates, obtained with the use of double categories.

Finally, [1, 2] may be viewed as preliminary steps towards our use of premonoidal structures. They are the first papers to integrate the indexed approach of [16, 45] with the fixpoint semantics in [23], and to show CLP as an instance of a standard categorical framework. Moreover, [1] also introduces the use of

premonoidal structures to model conjunctive goals. However, the fixpoint semantics construction in [1, 2] was quite limited compared with the present paper, since it only worked with free-goal logic programs. In order to overcome this limitation, the CLP presentations in these papers did not impose the regularity condition introduced in Equation 7.2, which makes dubious whether they are faithful interpretations of standard constraint logic programming. Moreover, [1, 2] did not give any practical means to build the syntactic LP doctrine from a finite specification, such as the display structures in our paper.

10. Conclusions

We have introduced a unified framework for declarative extensions to the syntax and semantics of Horn clause logic programming encompassing categorical logic, generalized unification, data types, constraints and state transformation using categories indexed over a finite product category. This framework builds a generic syntax, in which we can define terms, predicates, proofs and derivations over a general indexed premonoidal category. The framework also allows us to define an operational semantics as well as a categorical model theory, via doctrines, with respect to which the proof theory is sound and complete. Our formulation extends the so-called bottom-up approach to semantics defined by Kowalski and van Emden, which characterizes models as fixed points of a certain continuous operator, to our categorical setting.

Special cases of these language extensions are CLP [40], and in particular constraint logic programming using Saraswat's formulation [77], hyperdoctrinal constraints [68], and the sequent-based Nieva-Leach-Rodríguez Artalejo [53] approach, as well as logic programs with built-in data types [55]. Our framework has been able to naturally encompass these extensions. Their integration into an unique language with both constraints and abstract data types would be of great interest, as the resulting language would be very expressive, while remaining faithful to the declarative nature of logic programming.

A related framework has been used to include monad definitions [60], as well as First Order Hereditarily Harrop programs [24, 47] but without the full generality of the premonoidal doctrines defined in this paper. The next natural step is to broaden the current framework to include these extensions as well as to further develop static analysis tools [59] and abstract interpretation [19, 20, 15] in this setting, exploiting the bottom-up semantics we have given above. The internal logic of the categories in this paper is closer to linear than intuitionistic logic. It seems a natural step to flesh out the connections with logic programming using linear and other substructural logics [26, 27, 35, 62, 72] as well.

A fundamental development with such a general theory is to investigate how implementations of logic languages may be derived by descriptions of the syntactic LP doctrines. In other words, it would be of great interest to develop a skeletal interpreter and an abstract machine which, coupled with a description of $\mathcal{P}$ (such as, an actual algorithm to compute most general reduction pairs), yields a full interpreter for the corresponding language.

References

[1] G. Amato. *Sequent Calculi and Indexed Categories as a Foundation for Logic Programming*. PhD thesis, Università di Pisa, Dipartimento di Informatica, Mar. 2001.

[2] G. Amato and J. Lipton. Indexed categories and bottom-up semantics of logic programs. In R. Nieuwenhuis and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 8th International Conference, LPAR 2001 Havana, Cuba, December 3–7, 2001 Proceedings*, volume 2250 of *Lecture Notes in Artificial Intelligence*, pages 438–454. Springer, 2001.

[3] A. Asperti and G. Longo. *Categories, Types, and Structures: An Introduction to Category Theory for the Working Computer Scientist*. Foundations of Computing Series. The MIT Press, 1991.

[4] A. Asperti and S. Martini. Projections instead of variables. In G. Levi and M. Martelli, editors, *Logic Programming, Proceedings of the Sixth International Conference, Lisbon, Portugal, June 19-23, 1989*, pages 337–352. The MIT Press, 1989.

[5] E. S. Bainbridge, A. Scedrov, P. J. Freyd, and P. J. Scott. Functorial polymorphism. In G. P. Huet, editor, *Logical Foundations of Functional Programming*, chapter 14, pages 315–327. Addison Wesley, 1990.

[6] M. Baldoni, L. Giordano, and A. Martelli. A modal extension of logic programming: Modularity, beliefs and hypothetical reasoning. *Journal of Logic and Computation*, 8(5):597–635, 1998.

[7] 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.

[8] M. Barr and C. Wells. *Category Theory for Computing Science*. Prentice Hall, 1990.

[9] M. Barr and C. Wells. Toposes, theories and triples. *Reprints in Theory and Applications of Categories*, 12:1–287, 2005. Originally published by Springer-Verlag, 1985.

[10] R. Bird and O. de Moor. *Algebra of programming*. Prentice Hall, 1997.

[11] D. Cabeza, M. V. Hermenegildo, and J. Lipton. Hiord: A type-free higher-order logic programming language with predicate abstraction. In M. J. Maher, editor, *Advances in Computer Science - ASIAN 2004, Higher-Level Decision Making, 9th Asian Computing Science Conference, Chiang Mai, Thailand, December 8-10, 2004, Proceedings*, volume 3321 of *Lecture Notes in Computer Science*, pages 93–108. Springer, 2005.

[12] W. Chen, M. Kifer, and D. S. Warren. HiLog: A foundation for higher-order logic programming. *The Journal of Logic Programming*, 15(3):187–230, 1993.

[13] J. Cheney and C. Urban. α-Prolog: A logic programming language with names, binding and α-equivalence. In B. Demoen and V. Lifschitz, editors, *Logic Programming, 20th International Conference, ICLP 2004, Saint-Malo, France, September 6-10, 2004. Proceedings*, volume 3132 of *Lecture Notes in Computer Science*, pages 269–283. Springer, 2004.

[14] A. Colmerauer. An introduction to Prolog III. *Communications of the ACM*, 33(7):69–90, 1990.

[15] M. Comini, G. Levi, and M. C. Meo. A theory of observables for logic programs. *Information and Computation*, 169(1):23–80, 2001.

[16] A. Corradini and A. Asperti. A categorical model for logic programs: Indexed monoidal categories. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *Sematics: Foundations and Applications, REX Workshop, Beekbergen, The Netherlands, June 1-4, 1992, Proceedings*, volume 666 of *Lecture Notes in Computer Science*, pages 110–137. Springer, 1992.

[17] 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, Aug. 1992.

[18] A. Cortesi, G. Filé, and W. W. Winsborough. Prop revisited: Propositional formula as abstract domain for groundness analysis. In *Proceedings of the Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 322–327. IEEE Computer Society Press, 1991.

[19] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252. ACM Press, Jan. 1977.

[20] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 269–282. ACM Press, Jan. 1979.

[21] R. Diaconescu. *Category-based Semantics for Equational and Constraint Logic Programming.* PhD thesis, Oxford University, 1994.

[22] S. E. Finkelstein. *Tau Categories and Logic Programming.* PhD thesis, University of Pennsylvania, 1994.

[23] S. E. Finkelstein, P. J. Freyd, and J. Lipton. Logic programming in tau categories. In L. Pacholski and J. Tiuryn, editors, *Computer Science Logic 8th Workshop, CSL '94 Kazimierz, Poland, September 25–30, 1994. Selected Papers*, volume 933 of *Lecture Notes in Computer Science*, pages 249–263. Springer, 1995.

[24] S. E. Finkelstein, P. J. Freyd, and J. Lipton. A new framework for declarative programming. *Theoretical Computer Science*, 300(1–3):91–160, May 2003.

[25] P. J. Freyd and A. Scedrov. *Categories, Allegories*. North Holland, 1990.

[26] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987.

[27] J.-Y. Girard. On the unity of of logic. *Annals of Pure and Applied Logic*, 59(3):201–217, 1993.

[28] J. A. Goguen. What is unification? A categorical view of substitution, equation and solution. In M. Nivat and H. Aït-Kaci, editors, *Resolution of Equations in Algebraic Structures, Volume 1: Algebraic Techniques*, pages 217–261. Academic Press, 1989.

[29] J. A. Goguen and R. M. Burstall. Institutions: abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, 1992.

[30] J. A. Goguen and G. Malcolm. Hidden coinduction: behavioural correctness proofs for objects. *Mathematical Structures in Computer Science*, 9(3):287–319, 1999.

[31] J. A. Goguen and J. Meseguer. EqLog: Equality, types, and generic modules for logic programming. In D. DeGroot and G. Lindstrom, editors, *Logic Programming: Functions, Relations, and Equations*, pages 295–363. Prentice Hall, 1986.

[32] T. Hagino. A typed lambda calculus with categorical type constructors. In D. E. R. David H. Pitt, Axel Poign, editor, *Category Theory and Computer Science Edinburgh, U.K., September 7–9, 1987 Proceedings*, volume 283 of *Lecture Notes in Computer Science*, pages 140–157. Springer, 1987.

[33] M. Hanus. The integration of functions into logic programming: From theory to practice. *The Journal of Logic Programming*, 19 & 20:583–628, 1994.

[34] C. Hermida. *Fibrations, logical predicates and related topics*. PhD thesis, University of Edinburgh, 1993.

[35] J. S. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110(2):327–365, May 1994.

[36] G. Huet, editor. *Logical foundations of functional programming*. Addison-Wesley Longman Publishing, 1990.

[37] B. Jacobs. *Categorical Logic and Type Theory*, volume 141 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 1999.

[38] B. Jacobs. Exercises in coalgebraic specification. In *Algebraic and Coalgebraic Methods in the Mathematics of Program Construction International Summer School and Workshop Oxford, UK, April 10–14, 2000 Revised Lectures*, volume 2297 of *Lecture Notes in Computer Science*, pages 241–265. Springer, 2002.

[39] D. Jacobs and A. Langen. Accurate and efficient approximation of variable aliasing in logic programs. In E. L. Lusk and R. A. Overbeek, editors, *Logic Programming: The 1989 North American Conference*, pages 154–165. The MIT Press, 1989.

[40] J. Jaffar and M. J. Maher. Constraint logic programming: A survey. *The Journal of Logic Programming*, 19 & 20:503–581, 1994.

[41] J. Jaffar, S. Michaylov, P. J. Stuckey, and R. H. C. Yap. The CLP($\mathcal{R}$) language and system. *ACM Transactions on Programming Languages and Systems*, 14(3):339–395, 1992.

[42] J. Jaffar and P. J. Stuckey. Semantics of infinite tree logic programming. *Theoretical Computer Science*, 46(2-3):141–158, 1986.

[43] E. G. W. James W. Thatcher and J. B. Wright. Data type specification: Parameterization and the power of specification techniques. *ACM Transactions on Programming Languages and Systems*, 4(4):711–732, 1982.

[44] J. S. Jeavons. An alternative linear semantics for allowed logic programs. *Annals of Pure and Applied Logic*, 84(1):3–16, 1997.

[45] Y. Kinoshita and J. Power. A fibrational semantics for logic programs. In R. Dyckhoff, H. Herre, and P. Schroeder-Heister, editors, *Extensions of Logic Programming 5th International Workshop, ELP '96 Leipzig, Germany, March 28–30, 1996 Proceedings*, volume 1050 of *Lecture Notes in Artificial Intelligence*, pages 177–192. Springer, 1996.

[46] A. Kock and G. E. Reyes. Doctrines in categorical logic. In J. Barwise, editor, *Handbook of Mathematical Logic*, Studies in Logic and the Foundations of Mathematics, pages 283–313. North Holland, 1977.

[47] A. Krishnan. *Universal Quantifiers in Logic Programming via Indexed Categories*. PhD thesis, Wesleyan University, 2005.

[48] J. Lambek. From lambda calculus to cartesian closed categories. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 376–402. Academic Press, 1980.

[49] J. Lambek and P. J. Scott. *Introduction to Higher-Order Categorical Logic*, volume 7 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 1986.

[50] F. W. Lawvere. Adjointness in foundations. *Dialectica*, 23(3–4):281–296, 1969.

[51] F. W. Lawvere. Equality in hyperdoctrines and comprehension schema as an adjoint functor. In A. Heller, editor, *Proceedings of the New York Symposium on Applications of Categorical Algebra*, pages 1–14. American Mathematical Society, 1970.

[52] F. W. Lawvere. Functorial semantics of algebraic theories and some algebraic problems in the context of functorial semantics of algebraic theories. *Reprints in Theory and Applications of Categories*, 5:1–121, 2004. Reprint of Columbia University 1963 dissertation and Reports of the Midwest Category Seminar II, 1968, 41-61, Springer-Verlag, with new comments by the author added in 2004.

[53] J. Leach, S. Nieva, and M. Rodríguez-Artalejo. Constraint logic programming with hereditary Harrop formulas. *Theory and Practice of Logic Programming*, 1(4):409–445, 2001.

[54] J. J. Leifer and R. Milner. Deriving bisimulation congruences for reactive systems. In C. Palamidessi, editor, *CONCUR 2000 Concurrency Theory, 11th International Conference University Park, PA, USA, August 22–25, 2000 Proceedings*, volume 1877 of *Lecture Notes in Computer Science*, pages 243–258. Springer, 2000.

[55] J. Lipton and R. W. McGrail. Encapsulating data in logic programming via categorical constraints. In C. Palamidessi, H. Glaser, and K. Meinke, editors, *Principles of Declarative Programming, 10th International Symposium, PLILP98 Held Jointly with the 6th International Conference, ALP98 Pisa, Italy, September 16–18, 1998 Proceedings*, volume 1490 of *Lecture Notes in Computer Science*, pages 391–410. Springer, 1998.

[56] J. W. Lloyd. *Foundations of Logic Programming*. Springer, second edition, 1987.

[57] S. Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer, 1971.

[58] M. Makkai and G. E. Reyes. *First Order Categorical Logic, Model-Theoretical Methods in the Theory of Topoi and Related Categories*, volume 611 of *Lecture Notes in Mathematics*. Springer, 1977.

[59] K. Marriott, H. Søndergaard, and N. D. Jones. Denotational abstract interpretation of logic programs. *ACM Transactions on Programming Languages and Systems*, 16(3):607–648, 1994.

[60] R. W. McGrail. *Monads, Predicates and Categorical Logic Programming*. PhD thesis, Wesleyan University, 1999.

[61] D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.

[62] D. Miller. FORUM: a multiple-conclusion specification logic. *Theoretical Computer Science*, 165(1):201–232, 1996.

[63] D. Miller, F. Pfenning, G. Nadathur, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51(1–2):125–157, 1991.

[64] E. Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991.

[65] G. Nadathur. A proof procedure for the logic of hereditary Harrop formulas. *Journal of Automated Reasoning*, 11(1):115–145, Feb. 1993.

[66] G. Nadathur and D. Miller. Higher-order Horn clauses. *Journal of the ACM*, 37(4):777–814, Oct. 1990.

[67] F. Orejas, M. Navarro, and A. Sánchez. Algebraic implementation of abstract data types: A survey of concepts and new compositionality results. *Mathematical Structures in Computer Science*, 6(1):33–67, Feb. 1996.

[68] P. Panangaden, V. A. Saraswat, P. J. Scott, and R. A. G. Seely. A hyperdoctrinal view of concurrent constraint programming. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *Semantics: Foundations and Applications, REX Workshop Beekbergen, The Netherlands, June 1–4, 1992 Proceedings*, volume 666 of *Lecture Notes in Computer Science*, pages 457–476. Springer, 1993.

[69] S. L. Peyton Jones et al. *The Haskell 98 Language and Libraries: The Revised Report*, volume 13 of *Journal of Functional Programming*. Cambridge University Press, Jan 2003.

[70] D. H. Pitt, S. Abramsky, A. Poign, and D. E. Rydeheard, editors. *Category Theory and Computer Programming, Tutorial and Workshop, Guildford, U.K. September 16–20, 1985 Proceedings*, volume 240 of *Lecture Notes in Computer Science*. Springer, 1986.

[71] J. Power and E. Robinson. Premonoidal categories and notions of computation. *Mathematical Structures in Computer Science*, 7(5):453–468, Oct. 1997.

[72] D. J. Pym and J. Harland. A uniform proof-theoretic investigation of linear logic programming. *Journal of Logic and Computation*, 4(2):175–207, April 1994.

[73] H. Reichel. *Initial computability, algebraic specifications, and partial algebras*. Oxford International Series Of Monographs On Computer Science. Oxford University Press, 1987.

[74] M. H. Rogers and H. Abramson, editors. *Meta-programming in logic programming*. Logic Programming Series. The MIT Press, Oct. 1989.

[75] J. Rothe, H. Tews, and B. Jacobs. The coalgebraic class specification language CCSL. *Journal of Universal Computer Science*, 7(2):175–193, 2002.

[76] D. E. Rydeheard and R. M. Burstall. A categorical unification algorithm. In Pitt et al. [70], pages 493–505.

[77] V. A. Saraswat. The category of constraint systems is cartesian-closed. In *Proceedings of the Seventh Annual IEEE Symposium on Logic in Computer Science*, pages 341–345. IEEE Computer Society Press, June 1992.

[78] R. A. G. Seely. Hyperdoctrines, natural deduction and the Beck condition. *Mathematical Logic Quarterly*, 29(10):505–542, 1983.

[79] 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.

[80] E. G. Wagner. Categories, data types and imperative languages. In Pitt et al. [70], pages 143–162.

[81] E. G. Wagner. Algebraic specifications: Some old history and new thoughts. *Nordic Journal of Computing*, 9(4):373–404, 2002. Selected papers of the Thirteenth Nordic Workshop on Programming Theory (NWPT'01), October 10–12, 2001.

A. Proofs

A.1. Reduction Pairs and Unifiers

Proof of Theorem 5.15. Given a clause reduction pair $\langle r, fcl\rangle$ of $\mathbf{G} = A_1 \otimes \cdots \otimes A_n$, with the formal clause $fcl = \langle \mathbf{G}_a, s, cl, \mathbf{G}_b\rangle$ and $cl : \mathbf{Hd} \leftarrow \mathbf{Tl}$, we have that $\mathbf{G}_a \otimes s^\sharp \mathbf{Hd} \otimes \mathbf{G}_b = r^\sharp \mathbf{G}$. Since $r^\sharp \mathbf{G}$, $\mathbf{G}_a$ and $\mathbf{G}_b$ have unique decompositions as atomic goals and since reindexing preserves atomic goals, this is only possible if there is an $i \in \{1, \ldots, n\}$ such that $s^\sharp \mathbf{Hd} = r^\sharp A_i$, $\mathbf{G}_a = r^\sharp A_1 \otimes \cdots \otimes r^\sharp A_{i-1}$ and $\mathbf{G}_b = r^\sharp A_{i+1} \otimes \cdots \otimes r^\sharp A_n$. Therefore $\langle r, s\rangle$ is an unifier of A_i and $\mathbf{Hd}$.

Now assume $\langle r, s\rangle$ is maximal, and consider an arrow $t : \langle r, fcl\rangle \to \langle r', fcl'\rangle \in \mathsf{Red}_{\mathbf{G}}$. We have that $fcl = \langle \mathbf{G}'_a, s', cl, \mathbf{G}'_b\rangle$ where $\mathbf{G}'_a = r'^\sharp A_1 \otimes \cdots \otimes r'^\sharp A_{i-1}$, $\mathbf{G}'_b = r'^\sharp A_{i+1} \otimes \cdots \otimes r'^\sharp A_n$ and $ts' = s$. This means that t is also an arrow $\langle r, s\rangle \to \langle r', s'\rangle \in \mathsf{Unif}_{A,\mathbf{Hd}}$. By maximality of $\langle r, s\rangle$, there is an unique arrow $t' : \langle r', s'\rangle \to \langle r, s\rangle$ in $\mathsf{Unif}_{A,\mathbf{Hd}}$ such that $tt' = id$. But then, t' is also an arrow $\langle r', fcl'\rangle \to \langle r, fcl\rangle$. The proof that maximality of $\langle r, fcl\rangle$ implies maximality of $\langle r, s\rangle$ is similar. ∎

A.2. Building the Free Model

We want to relate the operational semantics based on categorical derivations and the model-theoretic semantics. We follow a standard methodology and we look for particular models of P, called *free models*, which enjoys a fundamental universal mapping property.

Definition A.1 (Free models) *Given a program P over $\mathcal{P}$, a model $M : \mathcal{P} \to \mathcal{Q}$ is* free *iff for every other model $M' : \mathcal{P} \to \mathcal{Q}'$ there exists an unique interpretation $N : \mathcal{Q} \to \mathcal{Q}'$ such that $M' = M \diamond N$.*

It is easy to prove by universality properties that, if M and M' are both free models for a program P in two different logic doctrines $\mathcal{Q}$ and $\mathcal{R}$, there is an isomorphism of indexed premonoidal categories $i : \mathcal{Q} \to \mathcal{R}$ such that $i(M(cl)) = N(cl)$. For these reason, we also speak of *the* free model.

We build from the LP doctrine $\mathcal{P}$ and the program P a particular free model $\mathcal{F}_P$ which is obtained by *freely adjoining* the clauses of P to the fibers of $\mathcal{P}$. In fact, categorical derivations are already obtained by adding clauses in P to arrows in $\mathcal{P}$, but in order to get a free model we need to get rid of redundant derivations and give them an indexed structure.

Definition A.2 (Flat derivations) *A categorical derivation d is called* flat *(on the fiber σ) when, for all steps $\langle r, f\rangle$ in d, it is the case that $r = id_\sigma$.*

If $d : \mathbf{G} \rightsquigarrow^* \mathbf{G}'$ is a flat derivation on the fiber σ and $k : \rho \to \sigma$ an arrow in $\mathbb{C}$, we define a new flat derivation $k^\sharp d : k^\sharp \mathbf{G} \rightsquigarrow^* k^\sharp \mathbf{G}'$ on the fiber ρ as follows:

$$k^\sharp(\epsilon_{\mathbf{G}}) = \epsilon_{k^\sharp \mathbf{G}} \ ,$$
$$k^\sharp(d \cdot \langle id_\sigma, f\rangle) = k^\sharp(d) \cdot \langle id_\rho, k^\sharp f\rangle \ .$$

It is easy to check that ${k_1}^\sharp {k_2}^\sharp(d) = (k_1 k_2)^\sharp(d)$. Moreover, given a flat derivation $\mathbf{G}_1 \overset{d}{\rightsquigarrow}^* \mathbf{G}_2$ on the fiber σ, and a goal $\mathbf{G} : \sigma$, it is possible to define a derivation $\mathbf{G} \mathbin{\bar{\otimes}} d : \mathbf{G} \otimes \mathbf{G}_1 \rightsquigarrow^* \mathbf{G} \otimes \mathbf{G}_2$, by induction on the length of d:

$$\begin{aligned}
&\mathbf{G} \mathbin{\bar{\otimes}} \epsilon_{\mathbf{G}_1} = \epsilon_{\mathbf{G} \otimes \mathbf{G}_1} \ , \\
&\mathbf{G} \mathbin{\bar{\otimes}} (d' \cdot \langle id, f \rangle) = (\mathbf{G} \mathbin{\bar{\otimes}} d') \cdot \langle id, \mathbf{G} \otimes f \rangle \ .
\end{aligned}$$

There is a similar operator $d \mathbin{\bar{\otimes}} \mathbf{G}$ whose definition is symmetrical. Note that $\top \mathbin{\bar{\otimes}} d = d \mathbin{\bar{\otimes}} \top = d$. We also define an operator "$\mathsf{flatten}$" on derivations which, from $d : \mathbf{G} \rightsquigarrow^* \mathbf{G}'$ with computed answer r, gives as a result a flat derivation $\mathsf{flatten}(d) : r^\sharp \mathbf{G} \rightsquigarrow^* \mathbf{G}'$:

$$\begin{aligned}
&\mathsf{flatten}(\epsilon_{\mathbf{G}}) = \epsilon_{\mathbf{G}} \ , \\
&\mathsf{flatten}(d \cdot \langle r, f \rangle) = r^\sharp(\mathsf{flatten}(d)) \cdot \langle id, f \rangle \ .
\end{aligned}$$

The operators $f^\sharp$ and $\mathbin{\bar{\otimes}}$ seem the building blocks of an indexed premonoidal category of flat derivations. However, we still need to remove some redundant derivations we obtain by the unrestricted application of the backchain rule.

Definition A.3 (Normal derivations) *A derivation is called* normal *when*

1. *there are no two consecutive steps with an arrow reduction pair;*
2. *there are no steps with an arrow reduction pair $\langle r, f \rangle$ where f is an identity arrow.*

We define an operator $\mathsf{normalize}$ that, from a derivation $d : \mathbf{G} \rightsquigarrow^* \mathbf{G}'$, builds a normal derivation $\mathsf{normalize}(d) : \mathbf{G} \rightsquigarrow^* \mathbf{G}'$. The operator is based on the following rewriting system on derivations:

$$\begin{aligned}
&\langle r_1, f_1 \rangle \cdot \langle r_2, f_2 \rangle \Longrightarrow \langle r_2 r_1, f_2 \diamond r_2^\sharp f_1 \rangle \qquad [\text{when } f_1, f_2 \text{ are arrows}] \ , \\
&\langle id_\sigma, id_{\mathbf{G}} \rangle \Longrightarrow \epsilon_{\mathbf{G}} \ .
\end{aligned}$$

Proposition A.4 *The rewriting system $\Longrightarrow$ is terminating and confluent.*

Proof. Termination is obvious, since each rewriting strictly reduces the length of the derivation. In order to prove confluence, consider all the subderivations where these rules overlap. We have the following cases:

- $\langle id_\sigma, id_{\mathbf{G}} \rangle \cdot \langle r, f \rangle$: we may apply either the first or the second rule, but in both case we obtain $\langle r, f \rangle$;
- $\langle r, f \rangle \cdot \langle id_\sigma, id_{\mathbf{G}} \rangle$: same as above;
- $\langle r_1, f_1 \rangle \cdot \langle r_2, f_2 \rangle \cdot \langle r_3, f_3 \rangle$: we may apply the first rule in two different ways, but in both cases another use of the same rule yields $\langle r_3 r_2 r_1, f_3 \diamond {r_3}^\sharp f_2 \diamond {r_3}^\sharp {r_2}^\sharp f_1 \rangle$.

This concludes the proof. ∎

Since $\Longrightarrow$ is terminating and confluent, we may define the operator $\mathsf{normalize}$ as follows:

$$\mathsf{normalize}(d) = d' \text{ such that } d \Longrightarrow^* d' \not\Longrightarrow \quad . \tag{A.1}$$

It enjoys some properties which will be useful later in the proofs:

- $\mathsf{normalize}$ is idempotent;
- if d is flat, $\mathsf{normalize}(d)$ is flat;
- if d is flat, $\mathsf{normalize}(t^\sharp(\mathsf{normalize}(d))) = \mathsf{normalize}(t^\sharp d)$.

We are now able to build the free model $\mathcal{F}_P$ for a program P over $\mathcal{P}$, by indexing normal flat derivations over their sort σ. We recall here Definition 5.23, so that this section is more self-contained.

Definition A.5 (The LP doctrine $\mathcal{F}_P$) *Given a program P over $\mathcal{P}$, we define the LP doctrine $\mathcal{F}_P : \mathbb{C}^{\mathrm{o}} \to \mathsf{Cat}$ as follows:*

- *$\mathcal{F}_P(\sigma)$ is the category of normal flat derivation of sort σ. More in detail:*
  - *$|\mathcal{F}_P(\sigma)| = |\mathcal{P}\sigma|$;*
  - *$\mathcal{F}_P(\sigma)(\mathbf{G}, \mathbf{G}')$ is the set of normal flat derivations from $\mathbf{G}'$ to $\mathbf{G}$;*
  - *$id_{\mathbf{G}} = \epsilon_{\mathbf{G}}$;*
  - *$dd' = \mathsf{normalize}(d' \cdot d)$;*
- *given $t : \sigma \to \rho$ in $\mathbb{C}$, $\mathcal{F}_P(t)$ is defined as follows:*
  - *on objects, $\mathcal{F}_P(t)(\mathbf{G}) = t^\sharp \mathbf{G}$;*
  - *on arrows, $\mathcal{F}_P(t)(d) = \mathsf{normalize}(t^\sharp d)$;*
- *for each $\sigma \in |\mathbb{C}|$, there is a strict premonoidal structure $(\otimes^{\mathcal{F}}_\sigma, \top^{\mathcal{F}}_\sigma)$ such that*
  - *$\top^{\mathcal{F}}_\sigma = \top^{\mathcal{P}}_\sigma$;*
  - *$\mathbf{G} \otimes^{\mathcal{F}}_\sigma d = \mathsf{normalize}(\mathbf{G} \bar{\otimes} d)$ and $d \otimes^{\mathcal{F}}_\sigma \mathbf{G} = \mathsf{normalize}(d \bar{\otimes} \mathbf{G})$.*

The proof that composition in $\mathcal{F}_P(\sigma)$ is associative and that $\epsilon_{\mathbf{G}}$ is the identity is trivial, given the confluence of $\Longrightarrow$. Moreover, if $t : \rho \to \sigma$, then $\mathcal{F}_P(t)$ is a functor from $\mathcal{F}_P(\sigma)$ to $\mathcal{F}_P(\rho)$, since

- $\mathcal{F}_P(t)(\epsilon_{\mathbf{G}}) = \mathsf{normalize}(t^\sharp \epsilon_{\mathbf{G}}) = \mathsf{normalize}(\epsilon_{t^\sharp \mathbf{G}}) = \epsilon_{t^\sharp \mathbf{G}}$;
- $\mathcal{F}_P(t)(d_1 \diamond d_2) = \mathcal{F}_P(t)(\mathsf{normalize}(d_2 \cdot d_1)) = \mathsf{normalize}(t^\sharp(d_2 \cdot d_1)) = \mathsf{normalize}(t^\sharp d_2 \cdot t^\sharp d_1)$. Since $\Longrightarrow$ is confluent, we have that $\mathsf{normalize}(t^\sharp d_2 \cdot t^\sharp d_1) = \mathsf{normalize}(\mathsf{normalize}(t^\sharp d_2) \cdot \mathsf{normalize}(t^\sharp d_1)) = \mathsf{normalize}(\mathcal{F}_P(t)(d_2) \cdot \mathcal{F}_P(t)(d_1)) = \mathcal{F}_P(t)(d_1) \diamond \mathcal{F}_P(t)(d_2)$.

Moreover, $\mathcal{F}_P$ is a functor, since

- $\mathcal{F}_P(id)(d) = \mathsf{normalize}(id^\sharp(d)) = \mathsf{normalize}(d) = d$ when d is normal;
- $\mathcal{F}_P(t_1)(\mathcal{F}_P(t_2)(d)) = \mathsf{normalize}(t_1{}^\sharp(\mathsf{normalize}(t_2{}^\sharp d)) = \mathsf{normalize}(t_1{}^\sharp t_2{}^\sharp d) =$

  $= \mathsf{normalize}((t_1 t_2)^\sharp(d)) = \mathcal{F}_P(t_1 t_2)(d)$.

For each σ, $\mathbf{G} \mathbin{\bar{\otimes}_\sigma} \_$ and $\_ \mathbin{\bar{\otimes}_\sigma} \mathbf{G}$ are functors. We just need to check that $\mathcal{F}_P(t)$ preserves the premonoidal structure. By induction on the length of d, we prove that $t^\sharp(\mathbf{G} \mathbin{\bar{\otimes}_\sigma} d) = t^\sharp\mathbf{G} \mathbin{\bar{\otimes}_\rho} t^\sharp d$:

- if $d = \epsilon_{\mathbf{G}'}$, then $t^\sharp(\mathbf{G} \mathbin{\bar{\otimes}} \epsilon_{\mathbf{G}'}) = t^\sharp(\epsilon_{\mathbf{G}\otimes\mathbf{G}'}) = \epsilon_{t^\sharp\mathbf{G}\otimes t^\sharp\mathbf{G}'} = t^\sharp\mathbf{G} \mathbin{\bar{\otimes}} t^\sharp\epsilon_{\mathbf{G}'}$;
- if $d = d' \cdot \langle id, f\rangle$, then $t^\sharp(\mathbf{G} \mathbin{\bar{\otimes}} d' \cdot \langle id, f\rangle) = t^\sharp((\mathbf{G} \mathbin{\bar{\otimes}} d') \cdot \langle id, \mathbf{G} \mathbin{\bar{\otimes}} f\rangle) = (t^\sharp\mathbf{G} \mathbin{\bar{\otimes}} t^\sharp d') \cdot \langle id, t^\sharp\mathbf{G} \mathbin{\bar{\otimes}} t^\sharp f\rangle = (t^\sharp\mathbf{G}) \mathbin{\bar{\otimes}} t^\sharp(d' \cdot \langle id, f\rangle)$.

Definition A.6 (The model F_P) *We define the model $F_P = (\llbracket \_ \rrbracket, \iota)$ of the program P over $\mathcal{P}$ into the LP doctrine $\mathcal{F}_P$ as follows:*

- $\iota(\mathbf{Hd} \xleftarrow{cl:\sigma} \mathbf{Tl}) = \mathbf{Hd} \xrightarrow{\langle id_\sigma, cl\rangle} \mathbf{Tl}$;
- $\llbracket \_ \rrbracket = \langle id_{\mathbb{C}}, \tau\rangle$;

where τ is defined as follows:

- $\tau_\sigma(\mathbf{G}) = \mathbf{G}$;
- $\tau_\sigma(f : \mathbf{G} \to \mathbf{G}') = \mathsf{normalize}(\mathbf{G}' \xrightarrow{\langle id_\sigma, f\rangle} \mathbf{G})$.

The $\mathsf{normalize}$ function in the definition above is needed for the case when $f = id_{\mathbf{G}}$. We need to check that F_P actually defines an interpretation. It is easy to check that τ_σ is a functor, and that it preserves the premonoidal structure. Moreover, it is a natural transformation, since

$$\tau_\rho(\mathcal{F}_P(t)(\mathbf{G} \xrightarrow{f} \mathbf{G}')) = \tau_\rho(t^\sharp\mathbf{G} \xrightarrow{t^\sharp f} t^\sharp\mathbf{G}') = \mathsf{normalize}(\langle id_\rho, t^\sharp f\rangle)$$

and

$$\mathcal{F}_P(t)(\tau_\sigma(f)) = \mathsf{normalize}(t^\sharp(\mathsf{normalize}(\langle id_\sigma, f\rangle))) = \mathsf{normalize}(t^\sharp(\langle id_\sigma, f\rangle)) = \mathsf{normalize}(\langle id_\rho, t^\sharp f\rangle) \ .$$

Theorem A.7 (The free model F_P) *F_P is a free model of P.*

Proof. Assume $M = (\llbracket \_ \rrbracket', \iota')$ is another model of P in the doctrine $\mathcal{Q}$. If $\llbracket \_ \rrbracket' = \langle F, \tau'\rangle$,we try to define an interpretation $N = (H, \tau'')$ from $\mathcal{F}_P$ to $\mathcal{Q}$ such that $\llbracket \_ \rrbracket \diamond N = \llbracket \_ \rrbracket'$ and $\iota \diamond N = \iota'$. Since $\llbracket \_ \rrbracket = (id_{\mathbb{C}}, \tau)$, it must be $H = F$. For the same reason $\tau''_\sigma(\mathbf{G}) = \tau'_\sigma(\mathbf{G})$ for each $\sigma \in |\mathbb{C}|$ and $\mathbf{G} \in |\mathcal{P}\sigma|$.

It remains to define τ''_σ on arrows. It is obvious that τ'' must satisfy the following constraints:

- $\tau''_\sigma(\mathsf{normalize}(\mathbf{G}' \xrightarrow{\langle id_\sigma, f\rangle} \mathbf{G})) = \tau'_\sigma(f)$ for any $f \in \mathcal{P}_\sigma$;
- $\tau''_\sigma(\mathbf{Hd} \xrightarrow{\langle id_\sigma, cl\rangle} \mathbf{Tl}) = \iota'(cl)$ for any $cl : \mathbf{Hd} \leftarrow \mathbf{Tl}$.

Now, consider a single derivation step $\langle id_\sigma, fcl\rangle$ where $fcl = \langle \mathbf{G}_a, s, cl, \mathbf{G}_b\rangle$, $s : \sigma \to \rho$ and $\mathbf{Hd} \xleftarrow{cl:\rho} \mathbf{Tl}$. It is equivalent to

$$\mathbf{G}_a \otimes^{\mathcal{F}}_\sigma \mathcal{F}(s)(\langle id_\rho, cl\rangle) \otimes^{\mathcal{F}}_\sigma \mathbf{G}_b \ . \tag{A.2}$$

Since τ''_σ should preserve reindexing and premonoidal structure, the value of $\tau''_\sigma(\langle id_\sigma, fcl\rangle)$ is forced by the definition of τ''_σ for the two cases above. Since all the arrows in $\mathcal{F}_P(\sigma)$ are obtained as composition of steps of the form $\langle id_\sigma, f\rangle$ and $\langle id_\sigma, fcl\rangle$, then τ''_σ is uniquely determined.

Note that uniqueness of τ'' is a consequence of the fact that $\mathcal{F}_P$ only contains normal flat derivations. Otherwise, given arrows $f_1 : \mathbf{G}_1 \to \mathbf{G}_2$ and $f_2 : \mathbf{G}_2 \to \mathbf{G}_3$ in the fiber $\mathcal{P}\sigma$, with $f_1 \neq id \neq f_2$, the fiber $\mathcal{F}_P(\sigma)$ would contain two different derivations $d = \mathbf{G}_3 \xrightarrow{\langle id, f_2\rangle} \mathbf{G}_2 \xrightarrow{\langle id, f_1\rangle} \mathbf{G}_1$ and $d' = \mathbf{G}_3 \xrightarrow{\langle id, f_1 f_2\rangle} \mathbf{G}_1$. The latter should be equal to the composition of $\mathbf{G}_2 \xrightarrow{\langle id, f_1\rangle} \mathbf{G}_1$ and $\mathbf{G}_3 \xrightarrow{\langle id, f_2\rangle} \mathbf{G}_2$ since $[\![\_]\!]$ should preserve composition of arrows, and mapped by τ'' to $\tau'(f_1 f_2)$. The value of $\tau''(d)$, however, may be chosen freely.

We should check that $\langle F, \tau''\rangle$ is indeed an interpretation of LP doctrines. It preserves composition and identities by construction. Moreover τ''_σ preserves premonoidal structures: for arrow reduction pairs it is a consequence of the fact that τ'_σ preserves premonoidal structures, while for clause reduction pairs preservation is given by construction. For the same reason, it commutes w.r.t. reindexing, i.e. $\tau''_\sigma(\mathcal{F}_P(t)(d)) = \mathcal{Q}_{Ft}(\tau''_\rho(d))$ for any $t : \sigma \to \rho$. ∎

From the fact that F_P is a free model, the following soundness and completeness results may be easily obtained.

Corollary A.8 (Soundness theorem) *Assume given a program P in $\mathcal{P}$, a goal $\mathbf{G}$ and a model $M = ([\![\_]\!], \iota) : \mathcal{P} \to \mathcal{Q}$. If d is a derivation of $\mathbf{G}$ to $\mathbf{G}'$ with answer t, there exists an arrow $[\![t^\sharp \mathbf{G}]\!] \leftarrow [\![\mathbf{G}']\!]$ in $\mathcal{Q}$.*

Proof. By Theorem A.7, there exists $N : \mathcal{F}_P \to \mathcal{Q}$ such that $M = F_P \diamond N$. Consider the normal flat derivation $d' = \mathsf{normalize}(\mathsf{flatten}(d)) : t^\sharp \mathbf{G} \rightsquigarrow^* \mathbf{G}'$. We know that d' is an arrow from $\mathbf{G}'$ to $t^\sharp \mathbf{G}$ in $\mathcal{F}_P(\sigma)$, where σ is the sort of $\mathbf{G}'$. Hence $N_\sigma(d)$ is an arrow from $N(\mathcal{F}_P(\mathbf{G}')) = [\![\mathbf{G}']\!]$ to $N(\mathcal{F}_P(t^\sharp \mathbf{G})) = [\![t^\sharp \mathbf{G}]\!]$. ∎

Corollary A.9 (Completeness theorem) *Assume given a program P in $\mathcal{P}$ and goals $\mathbf{G}$, $\mathbf{G}'$ of sort σ. If $M : \mathcal{P} \to \mathcal{Q}$ is a free model of P and there is an arrow $f : M(\mathbf{G}') \to M(\mathbf{G})$ in the fiber $M(\sigma)$, then there is a normal flat derivation $\mathbf{G} \overset{d}{\rightsquigarrow}{}^* \mathbf{G}'$.*

Proof. Since M is a free model, we can write $F_P = M \diamond N$ for some interpretation $N : \mathcal{Q} \to \mathcal{F}_P$. Hence, $N(f)$ is an arrow from $N(M(\mathbf{G}')) = \mathbf{G}'$ to $N(M(\mathbf{G})) = \mathbf{G}$. By definition of F_P, we have that $N(f)$ is a normal flat derivation $d : \mathbf{G}' \rightsquigarrow^* \mathbf{G}$. ∎

In particular, these results imply Theorem 5.22.

Proof of Theorem 5.22. In the hypotheses of the theorem, assume there is a categorical derivation $d : \mathbf{G} \rightsquigarrow^* \mathbf{G}'$ with answer $t : \sigma \to \rho$. By Corollary A.8, for every model M of P there in an arrow $M(t^\sharp \mathbf{G}) \leftarrow M(\mathbf{G}')$ in $M(\sigma)$. For the converse, assume that for every model M of P there in an arrow $f : M(t^\sharp \mathbf{G}) \leftarrow M(\mathbf{G}')$ in $M(\sigma)$. Then, the arrow also exists when $M = F_P$. By Corollary A.9 there is a normal flat derivation $d : t^\sharp \mathbf{G} \rightsquigarrow^* \mathbf{G}'$, hence $\langle t, id_\sigma \rangle \cdot d$ is a derivation $\mathbf{G} \rightsquigarrow^* \mathbf{G}'$ with answer t. ∎

A.3. Weak Completeness

In order to prove weak completeness of $[\![\_]\!]^\omega$, we introduce the following lemma, which may be viewed as a generalization of one of the conditions in the definition of well-behaved units.

Lemma A.10 *Assume given a semantic LP doctrine $\mathcal{Q}$ with well-behaved units, $I = \{1, \ldots, n\}$, $J \subseteq I$ and, for each $j \in J$, an arrow t_j in the base category of $\mathcal{Q}$. Moreover, assume given a family of objects $B_1, \ldots, B_n$ in the fiber $\mathcal{Q}\sigma$ such that, for each $j \in J$, $B_j = \exists_{t_j} X_j$. Then if $\top \leq \otimes_{i \in I} B_i$, there is a family of arrows $\{s_j\}_{j \in J}$ such that $s_j t_j = id$ and $\top \leq \otimes_{i \in I} C_i$ where $C_i = B_i$ if $i \notin J$ and $B_i = {s_i}^\sharp X_i$ if $i \in J$.*

Proof. The proof proceeds by induction on n. If $n = 0$ there is nothing to prove. If $n > 0$, take a $j \in J$, and by Frobenius reciprocity we get $\top \leq \exists_{t_j} \otimes_{i \in I} D_i$ where $D_i = t_j^\sharp B_i$ if $i \neq j$ and $D_i = X_j$ if $i = j$. Since $\mathcal{Q}$ has well-behaved units, then $\top \leq {s_j}^\sharp \otimes_{i \in I} D_i = \otimes_{i \in I} {s_j}^\sharp D_i = \otimes_{i \in I} E_i$ where $E_i = {s_j}^\sharp {t_j}^\sharp B_i = B_i$ if $i \neq j$ and $E_j = {s_j}^\sharp X_j$. Then, consider $J' = J \setminus \{j\}$, and by inductive hypothesis we have that $\top \leq \otimes_{i \in I} C_i$ where $C_i = E_i = B_i$ if $i \notin J$, $C_i = {s_i}^\sharp X_i$ if $i \in J' = J \setminus \{j\}$ and $C_j = {s_j}^\sharp X_j$. But this is exactly what we were looking for, hence the lemma is proved. ∎

Lemma A.11 *Given a weakly complete interpretation $[\![\_]\!]$ of an LP doctrine $\mathcal{P}$ into a semantic LP doctrine with well-behaved units $\mathcal{Q}$ and a program P over $\mathcal{P}$, then $\mathbb{E}_P([\![\_]\!])$ is weakly complete.*

Proof. Let us denote $\mathbb{E}_P([\![\_]\!])$ with $[\![\_]\!]'$ and assume $\top \leq [\![\mathbf{G}]\!]'$. If $\mathbf{G} = \top$, then $\epsilon_\top$ is a trivial successful derivation of $\top$ with identity answer. Otherwise $\mathbf{G} = A_1 \otimes \cdots \otimes A_n$, possibly with $n = 1$, and $\top \leq [\![\mathbf{G}]\!]'$ actually means $\top \leq [\![A_1]\!]' \otimes \cdots \otimes [\![A_n]\!]'$. By additivity of $\otimes$ and since $\mathcal{Q}$ has well-behaved units, it means that there are $J \subseteq \{1, \ldots, n\}$ and, for each $j \in J$, a reducer $\langle t_j, f_j, fcl_j \rangle$ of A_j into $\mathbf{Tl}_j$ such that $\top \leq \otimes_{i=1}^n B_i$, where $B_i = \exists_{t_i} [\![\mathbf{Tl}_i]\!]$ if $i \in J$, while

$B_i = [\![A_i]\!]$ otherwise. Without loss of generality, assume $J = \{j, \ldots, n\}$ for some $j \in I$. Then

$$\top \leq [\![A_1]\!] \otimes \cdots \otimes [\![A_{j-1}]\!] \otimes \exists_{t_j} [\![\mathbf{Tl}_j]\!] \otimes \cdots \otimes \exists_{t_n} [\![\mathbf{Tl}_n]\!] \ .$$

By Lemma A.10, we have that for each j there exists s_j such that $s_j t_j = id$ and

$$\top \leq [\![A_1 \otimes \cdots \otimes A_{j-1} \otimes {s_j}^\sharp \mathbf{Tl}_j \otimes \cdots \otimes {s_n}^\sharp \mathbf{Tl}_n]\!] \ .$$

Since $[\![\_]\!]$ is weakly complete, $A_1 \otimes \cdots \otimes A_{j-1} \otimes {s_j}^\sharp \mathbf{Tl}_j \otimes \cdots \otimes {s_n}^\sharp \mathbf{Tl}_n$ has a successful derivation d' with identity answer. Moreover, there are derivations d'_j for each j such that

$$d'_j = \mathbf{G}_j \otimes A_j \otimes \mathbf{G}'_j \xrightarrow{\langle s_j t_j, \mathbf{G}_j \otimes {s_j}^\sharp(f_j) \otimes \mathbf{G}'_j, \mathbf{G}_j \otimes {s_j}^\sharp(fcl_j) \otimes \mathbf{G}'_j \rangle} \otimes \mathbf{G}_j \otimes {s_j}^\sharp \mathbf{Tl}_j \otimes \mathbf{G}'_j \ , \tag{A.3}$$

where $\mathbf{G}_j = A_1 \otimes \cdots \otimes A_{j-1}$ and $\mathbf{G}'_j = {s_{j+1}}^\sharp \mathbf{Tl}_{j+1} \otimes \cdots \otimes {s_n}^\sharp \mathbf{Tl}_n$. Note that, by the choice of s_j, $\mathsf{answer}(d'_j) = s_j t_j = id$. We may concatenate all these derivations to obtain

$$d = d'_n \cdot d'_{n-1} \cdot \ldots \cdot d'_j \cdot d' \ ,$$

which is a successful derivation of $\mathbf{G}$ with identity answer. ∎

Finally, the proof of the main theorem is just a simple application of the previous Lemma.

Proof of Theorem 5.40. If $\top \leq [\![\mathbf{G}]\!]^\omega$, since units in $\mathcal{Q}$ are well-behaved, there exists $n \in \mathbb{N}$ such that $\top \leq [\![\mathbf{G}]\!]^n$. By the previous lemma we have that $[\![\mathbf{G}]\!]^n$ is weakly complete, hence there exists a derivation $d : \mathbf{G} \rightsquigarrow^* \top$ with $\mathsf{answer}(d) = id$. ∎

A.4. Yoneda Semantics

Proof of Theorem 6.2. Let us check all the properties which characterize the semantic LP doctrines:

1. $\mathcal{Y}_{\mathbb{C}}(\sigma)$ is a complete lattice.
2. $\mathcal{Y}_{\mathbb{C}}(t)$ is a complete join-morphism. Given $t : \rho \to \sigma$ and $\{X_i\}_{i \in I}$ a family of canonical subobjects of $Hom(\_, \sigma)$, we have

$$\begin{aligned} \mathcal{Y}_{\mathbb{C}}(t)\left(\bigcup_{i \in I} X_i\right) &= \{r \in \mathbb{C} \mid rt \in \bigcup_{i \in I} X_i\} = \{r \in \mathbb{C} \mid \exists i \in I . rt \in X_i\} \\ &= \bigcup_{i \in I} \{r \in \mathbb{C} \mid rt \in X_i\} = \bigcup_{i \in I} \mathcal{Y}_{\mathbb{C}}(t)(X_i) \ . \end{aligned} \tag{A.4}$$

3. $\otimes_\sigma$ is a complete join-morphism, since $\mathcal{Y}_{\mathbb{C}}(\sigma)$ is distributive and $\otimes_\sigma$ is the meet of the lattice.
4. $\mathcal{Y}_{\mathbb{C}}(t)$ has a well-known left adjoint $\exists_t$ such that, if $t : \rho \to \sigma$ and $X \subseteq Hom(\_, \rho)$, then $\exists_t X = \{rt \mid r \in X\}$.

5. The extended Beck condition is satisfied. Assuming $r_1 \in r^\sharp \exists_s X$, we have $r_1 r \in \exists_s X$ and $r_1 r = s_1 s$ for some $s_1 \in X$. Therefore $id \in {s_1}^\sharp X$ and $r_1 \in \exists_{r_1} {s_1}^\sharp X$.
6. Frobenius reciprocity holds. Given $r : \rho \to \sigma$, $X_2 \subseteq \mathit{Hom}(\_, \sigma)$ and $X_1 \subseteq \mathit{Hom}(\_, \rho)$, we have $\exists_r (r^\sharp X_2 \cap X_1) = \exists_r \{t \mid tr \in X_2 \wedge t \in X_1\} = \{tr \mid tr \in X_2 \wedge t \in X_1\} = \{t' \mid t' \in X_2 \wedge t' = tr \wedge t \in X_1\} = X_2 \cap \exists_r X_1$.

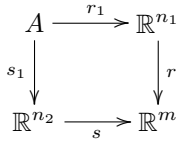
This concludes the proof of the theorem. ∎

A.5. Constraint Systems

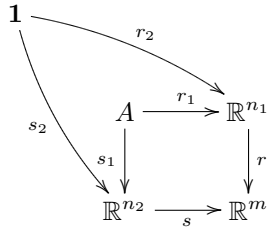
Proposition A.12 *$\mathcal{R}$ is a constraint system.*

Proof. We check the several conditions in the definition of constraint system:

- $\wp(\mathbb{R}^n)$ is a complete lattice, hence a bounded meet-preorder.
- $\exists_t$ is left adjoint to $t^\sharp$, since $\exists_t t^\sharp X = t(t^{-1}(X)) \subseteq X$ and $t^{-1}(t(X)) \supseteq X$.
- If the following diagram is a pullback



  and $X \in \mathcal{R}(\mathbb{R}^{n_1})$, we already know that $\exists_{s_1} {r_1}^\sharp X \subseteq s^\sharp \exists_r X$. We need to prove the converse inequality. Given $v \in s^\sharp \exists_r (X)$, there is $x \in X$ such that $s(v) = r(x)$. We want to prove that there is $z \in A$ such that $r_1(z) = x$ and $s_1(z) = v$, since this would immediately imply $s_1(r_1^{-1}(X)) \ni s_1(z) = v$. Assume this is not true, and take $A = \mathbb{R}^0 = \mathbf{1}$, $r_2(\cdot) = x$ and $s_2(\cdot) = v$. Then we have the following commutative diagram:



  However, the cone $(1, r_2, s_2)$ does not factor trough (A, r_1, s_1), hence (A, r_1, s_1) is not a pullback.

- Frobenius Reciprocity holds since $r(r^{-1}X \cap Y) = \{r(y) \mid y \in r^{-1}(X) \wedge y \in Y\} = \{r(y) \mid r(y) \in X \wedge y \in Y\} = \{x \mid x \in X \wedge x = r(y) \wedge y \in Y\} = X \cap r(Y)$.

This concludes the proof. Note that we have not used any special property of the set $\mathbb{R}$, so the proof is essentially the same for any other indexed category built in the same way by an appropriate subcategory of Set. ∎

Theorem A.13 *Given a constraint system $\mathcal{C}$, a constraint c over ρ, $r_1, r_2 : \rho \to \sigma$ and $t : \sigma \to \sigma'$, if $c \leq r_1 \approx r_2$ then $c \leq r_1 t \approx r_2 t$.*

Proof. Consider the following commutative diagram:

$$\begin{array}{ccc} \sigma & \xrightarrow{\quad t \quad} & \sigma' \\ {\scriptstyle \Delta_\sigma}\downarrow & & \downarrow{\scriptstyle \Delta_{\sigma'}} \\ \sigma \times \sigma & \xrightarrow[t \times t]{} & \sigma \times \sigma' \end{array} \tag{A.5}$$

We have already shown that $\exists_{\Delta_\sigma} t^\sharp \leq (t \times t)^\sharp \exists_{\Delta_{\sigma'}}$. By hypothesis we know that $c \leq (r_1 \approx r_2) = \langle r_1, r_2 \rangle^\sharp \exists_{\Delta_\sigma} \mathsf{true}_\sigma = \langle r_1, r_2 \rangle^\sharp \exists_{\Delta_\sigma} t^\sharp \mathsf{true}_{\sigma'}$. By the previous inequality we have $c \leq \langle r_1, r_2 \rangle^\sharp (t \times t)^\sharp \exists_{\Delta_{\sigma'}} \mathsf{true}_{\sigma'} = (r_1 t \approx r_2 t)$. ∎

A.6. Correspondence between CLP and the sequent calculus for constraints

In the following, given a constraint $c : \mathcal{C}(n)$ and a substitution $\theta : m \to n$, we will write $\theta^\sharp(c)$ in categorical derivations, where we want to stress the fact that a constraint is an object in a specific fiber, while we will use the more conventional notation $c\theta$ in proof trees, when the fibered structure of the constraints is not important. We will generally drop the superscript $\mathcal{C}$ from $\exists^{\mathcal{C}}_\theta$, since there is no confusion between the symbol $\exists$ and the operator $\exists_\theta$.

Note that, given substitutions $\theta, \theta' : n \to m$, there are two constraints which represent their equality. One is the constraint $\overline{\theta} \sim \overline{\theta'}$, which is a short form for $\theta(v_1) \sim \theta'(v_1) \wedge \ldots \theta(v_m) \sim \theta'(v_m)$, and the other is $\theta \approx \theta'$, according to the definition of $\_ \approx \_$ given in Section 7.1, both in the fiber $\mathcal{D}(n)$. It is possible to show that they are isomorphic.

To prove correspondence between CLP and $\mathcal{IC}$ we use the concept of *derived inference rule*. We say that

$$\frac{\Delta_1; \Gamma_1 \vdash G_1 \qquad \Delta_2; \Gamma_2 \vdash G_2}{\Delta; \Gamma \vdash G}$$

is a derived inference rule in $\mathcal{IC}$ if there is a proof tree Π for the sequent $\Delta; \Gamma \vdash G$ such that, for each leaf $\Delta'; \Gamma' \vdash G'$ in Π,there is $i \in \{1, 2\}$ and a renaming ρ such that $\Delta' = \Delta_i \rho$, $G' = G_i \rho$ and $\Gamma' \vdash_{\mathcal{C}} \Gamma_i \rho$. This means that, if Π_i is a proof of $\Delta_i; \Gamma_i \vdash G_i$, this may be turned into a proof of $\Delta'; \Gamma' \vdash G'$ by renaming free variables and replacing $\Gamma_i \rho$ with Γ'.[5] The definition extends immediately to rules with different number of premises.

[5] This is not true in the presence of the rule $(\forall_R)$, which is only used for Hereditary Harrop formulas.

If Π is a tree built from standard and derived inference rules, there is a derived inference rule with the same root and leafs of Π. Moreover, if Π has no leafs, there is a proof tree with no leafs for the root sequent of Π. This means that we can use derived inference rules as they were standard inference rules, without compromising the logical meaning of proof trees.

Given a sequence $\vec{x} = x_1, \ldots, x_n$ of variables, we denote with $\forall_{\vec{x}} D$ the formula $\forall_{x_1} \cdots \forall_{x_n} D$. The same notation is used for existential quantification.

Lemma A.14 *The following is a derived inference rule in $\mathcal{IC}$:*

$$\frac{\Delta, D[\vec{x}/\vec{y}]; \Gamma, c \vdash G \qquad \Gamma \vdash_{\mathcal{C}} \exists_{\vec{y}} c}{\Delta, \forall_{\vec{x}} D; \Gamma \vdash G} \; (\vec{\forall}_L)$$

where none of the variable in $\vec{y}$ appears free in the sequent of the conclusion.

Proof. Assume $\Gamma \vdash_{\mathcal{C}} \exists_{\vec{y}} c$. If $\vec{x} = x_1, \ldots, x_n$ and $\vec{y} = y_1, \ldots, y_n$, consider the following proof tree, obtained by repeated applications of the $(\forall_L)$ and (C_R) inference rules:

$$\dfrac{\dfrac{\dfrac{\Delta, D[\vec{x}/\vec{y}]; \Gamma_{n+1} \vdash G \qquad \overline{\Gamma_n \vdash_{\mathcal{C}} \exists_{y_n} c}}{\vdots \qquad \overline{\Gamma_2 \vdash_{\mathcal{C}} \exists_{y_2,\ldots,y_n} c}}}{\Delta, \forall_{x_2,\ldots,x_n} D[x_1/y_1]; \Gamma_2 \vdash G} \qquad \overline{\Gamma_1 \vdash_{\mathcal{C}} \exists_{y_1,\ldots,y_n} c}}{\Delta, \forall_{x_1,\ldots,x_n} D; \Gamma_1 \vdash G}$$

where $\Gamma_1 = \Gamma$ and $\Gamma_{i+1} = \Gamma_i \cup \{\exists_{y_{i+1},\ldots,y_n} c\}$. This is a valid proof tree since $\Gamma_1 \vdash_{\mathcal{C}} \exists_{y_1,\ldots,y_n} c$ by hypothesis, and $\exists_{y_i,\ldots,y_n} c \in \Gamma_i$ for each $i \geq 2$. Note that $c \in \Gamma_{n+1}$, hence $\Gamma_{n+1} \vdash_{\mathcal{C}} \Gamma, c$. This proves that $(\vec{\forall}_L)$ is a derived inference rule. ∎

Lemma A.15 *Given an atomic program P in $\mathcal{P}_{\mathbb{J},\mathcal{C}}$, with $\delta : \mathbb{J} \to \mathbb{S}_\Sigma$, assume there is a backchain step with a clause reduction pair*

$$\mathbf{G} : \langle n, c \rangle \xrightarrow{\langle \theta, \theta'^\sharp \mathbf{G}_a \otimes \theta'^\sharp cl \otimes \theta^\sharp \mathbf{G}_b \rangle} \theta^\sharp \mathbf{G}_a, \theta'^\sharp \mathbf{Tl}, \theta^\sharp \mathbf{G}_b : \langle m, c'' \rangle \;\; ,$$

such that $\mathbf{G} = A_1, \ldots, A_k$, $\mathbf{G}_a = A_1, \ldots, A_{i-1}$, $\mathbf{G}_b = A_{i+1}, \ldots, A_k$ and $\mathbf{Hd} \xleftarrow{cl:\langle n', c' \rangle} \mathbf{Tl}$. Then, the following is a derived rule in $\mathcal{IC}$:

$$\frac{\{\overline{P}; \exists_\theta c'' \vdash \overline{A_j}\}_{j \in \{1,\ldots,i-1\}} \qquad \overline{P}; \exists_{\theta'} c'' \vdash \overline{\mathbf{Tl}} \qquad \{\overline{P}; \exists_\theta c'' \vdash \overline{A_j}\}_{j \in \{i+1,\ldots,k\}}}{\overline{P}; \exists_\theta c'' \vdash \overline{\mathbf{G}}}$$

Proof. Let $\pi_1 : n + n' \to n = \{v_1/v_1, \ldots, v_n/v_n\}$ and $\pi_2 : n + n' \to n' = \{v_1/v_{n+1}, \ldots, v_{n'}/v_{n+n'}\}$ be the projections from $n + n'$ to n and n' respectively, while $\langle \theta, \theta' \rangle$ is the unique substitution $\theta'' = \{v_1/\theta(v_1), \ldots, v_n/\theta(v_2), v_{n+1}/\theta'(v_1), \ldots, v_{n+n'}/\theta'(v_{n'})\} : m \to n+n'$ such that $\theta'' \diamond \pi_1 = \theta$ and $\theta'' \diamond \pi_2 = \theta'$. Note that, if we look at substitutions and constraints as syntactic objects, then $\eta\pi_1 = \eta$ for any $\eta : m' \to n + n'$ and $c\pi_1 = c$ for any $c \in \mathcal{C}(n)$. However,

from the categorical point of view, $\eta \diamond \pi_1$ has a different target w.r.t. η, while ${\pi_1}^\sharp c$ and c are constraints in different fibers. Moreover, assume without loss of generality that $\overline{A} = p(\bar{t})$ and $\overline{\mathbf{Hd}} = p(\overline{t'})$.

Let us denote with Π_1 the following proof tree:

$$\dfrac{\dfrac{\dfrac{\overline{\Gamma \vdash_{\mathcal{C}} c'\pi_2}}{\overline{P}; \Gamma \vdash c'\pi_2} C_R \qquad \overline{P}; \Gamma \vdash \overline{\mathbf{Tl}}\pi_2}{\overline{P}; \Gamma \vdash (c' \wedge \overline{\mathbf{Tl}})\pi_2} (\wedge_R) \qquad \dfrac{\overline{\Gamma \vdash_{\mathcal{C}} \bar{t} \sim \overline{t'}\pi_2}}{\overline{P}, \overline{\mathbf{Hd}}\pi_2; \Gamma \vdash p(\bar{t})} \mathit{Atom}}{\overline{P}, ((c' \wedge \overline{\mathbf{Tl}}) \Rightarrow \overline{\mathbf{Hd}})\pi_2; \Gamma \vdash p(\bar{t})} (\Rightarrow_L)$$

where $\Gamma = \{\exists_\theta c'', \exists_{\langle \theta, \theta' \rangle} c''\}$. We need to show that $\Gamma \vdash_{\mathcal{C}} \bar{t} \approx \overline{t'}\pi_2$ and $\Gamma \vdash_{\mathcal{C}} c'\pi_2$. By definition of the backchain step, we know that

$$c'' \leq (\theta \diamond t) \approx (\theta' \diamond t') = \langle \theta \diamond t, \theta' \diamond t' \rangle^\sharp \exists_\Delta \mathsf{true} = \\ \langle \theta, \theta' \rangle^\sharp \langle \pi_1 \diamond t, \pi_2 \diamond t' \rangle^\sharp \exists_\Delta \mathsf{true} = \langle \theta, \theta' \rangle^\sharp ((\pi_1 \diamond t) \approx (\pi_2 \diamond t')) \ , \quad \text{(A.6)}$$

hence $\exists_{\langle \theta, \theta' \rangle} c'' \leq (\pi_1 \diamond t) \approx (\pi_2 \diamond t')$. Therefore, $\Gamma \vdash_{\mathcal{C}} \bar{t}\pi_1 \sim \overline{t'}\pi_2$ and since syntactically $\bar{t}\pi_1 = \bar{t}$, then $\Gamma \vdash_{\mathcal{C}} \bar{t} \sim \overline{t'}\pi_2$. Moreover, we know $c'' \leq {\theta'}^\sharp c' = (\langle \theta, \theta' \rangle \diamond \pi_2)^\sharp c' = \langle \theta, \theta' \rangle^\sharp {\pi_2}^\sharp c'$. Therefore, $\exists_{\langle \theta, \theta' \rangle} c'' \leq {\pi_2}^\sharp c'$, hence $\Gamma \vdash_{\mathcal{C}} c'\pi_2$.

Given $\vec{V} = v_1, \ldots, v_{n'}$, let us denote with Π_2 the following proof tree, where we use the derived inference rule introduced in Lemma A.14:

$$\dfrac{\begin{array}{c}\Pi_1 \\ \overline{P}, ((c' \wedge \overline{\mathbf{Tl}}) \Rightarrow \overline{\mathbf{Hd}})\pi_2; \Gamma \vdash p(\bar{t})\end{array} \qquad \overline{\exists_\theta c'' \vdash_{\mathcal{C}} \exists_{\vec{V}} \exists_{\langle \theta, \theta' \rangle} c''}}{\overline{P}, \forall_{\vec{V}} ((c' \wedge \overline{\mathbf{Tl}}) \Rightarrow \overline{\mathbf{Hd}}); \exists_\theta c'' \vdash p(\bar{t})} (\vec{\forall}_L)$$

We need to prove $\exists_\theta c'' \leq \exists_{\vec{V}} \exists_{\langle \theta, \theta' \rangle} c''$. We know that, when $c \in \mathcal{C}(n + n')$, then $\exists_{\vec{V}} c$ is isomorphic to $\exists_{\pi_1} c$. Therefore, it is enough to prove that $\exists_\theta c'' \leq \exists_{\pi_1} \exists_{\langle \theta, \theta' \rangle} c'' = \exists_\theta c''$ which is trivially true.

Finally, by using Π_2 and repeated applications of the $(\wedge_R)$ rule, we get a proof tree with root $\overline{P}; \exists_\theta c'' \vdash \overline{\mathbf{G}}$ and leafs $\overline{P}; \exists_\theta c'' \vdash \overline{A_i}$ for $i = \{1, \ldots, j-1\} \cup \{j+1, \ldots, k\}$ and $\overline{P}; \Gamma \vdash \overline{\mathbf{Tl}}\pi_2$. If we consider the renaming $\rho = \pi_2 \cup \{v_{n'+1}/v_1, \ldots . v_{n'+n}/v_n\}$, we have that $\overline{P}\rho = \overline{P}$, $\overline{\mathbf{Tl}}\rho = \overline{\mathbf{Tl}}\pi_2$. Moreover, $(\exists_{\theta'} c'')\rho = {\pi_2}^\sharp \exists_{\theta'} c'' \geq \exists_{\langle \theta, \theta' \rangle} c''$, hence $\Gamma \vdash_{\mathcal{C}} \exists_{\theta'} c''\rho$. Therefore, we have a derived inference rule. ∎

Proof of Theorem 7.9. We prove a slightly more general statement, namely that if there is a categorical derivation

$$\mathbf{G}\eta : \langle \sigma, c_1 \rangle \overset{d}{\rightsquigarrow}^* \top_\rho : \langle \rho, c_2 \rangle \ ,$$

and $\theta = \mathsf{answer}(d)$, then there exists an proof in $\mathcal{IC}$ of the sequent

$$\cdot \quad \overline{P}; \exists_{\theta \diamond \eta} c_2 \vdash \overline{\mathbf{G}}$$

The proof is by induction on the number n of steps in the derivation d. If $n = 0$ then $\mathbf{G} = \top_\rho$, i.e. the empty sequence, $\mathsf{answer}(d) = id_\rho$ and the proof follows by axiom (C_R) since $\overline{\mathbf{G}} = \mathsf{true}$. Otherwise, assuming a clause reduction pair is used in the first step of d, we have:

$$d = \mathbf{G}\eta : \langle n, c_1\rangle \xrightarrow{\langle \theta_1, \theta_1{}^\sharp \mathbf{G}_a \otimes \theta_1'{}^\sharp cl \otimes \theta_1{}^\sharp \mathbf{G}_b\rangle} \rightsquigarrow \mathbf{G}' : \langle m, c''\rangle \cdot d' \ ,$$

where $\mathbf{G} = A_1, \ldots, A_k$, $\mathbf{G}_a = A_1\eta, \ldots, A_{i-1}\eta$, $\mathbf{G}_b = A_{i+1}\eta, \ldots, A_k\eta$, $cl : \mathbf{Hd} \leftarrow \mathbf{Tl}$, $\mathbf{G}' = A_1\eta\theta_1, \ldots, A_{i-1}\eta\theta_1, \mathbf{Tl}\theta_1', A_{i+1}\eta\theta_1, \ldots, A_k\eta\theta_1$ and

$$d' = \mathbf{G}' : \langle m, c''\rangle \rightsquigarrow^* \top_\rho : \langle \rho, c_2\rangle \ ,$$

with $\mathsf{answer}(d') = \theta_2$ and $\theta_2 \diamond \theta_1 = \theta$. We may prove that it is possible to extract from d' proofs $d_i : A_i\eta\theta_1 : \langle m, c''\rangle \rightsquigarrow^* \top_\rho : \langle \rho, c_2\rangle$ with $\mathsf{answer}(d_i) = \theta_2$ for each $i \in \{1, \ldots, i-1\} \cup \{i+1, \ldots, k\}$ and a proof $d_* : \mathbf{Tl}\theta_1' \rightsquigarrow^* \top_\rho : \langle \rho, c_2\rangle$ with $\mathsf{answer}(d_*) = \theta_2$. By inductive hypothesis, there are proof trees Π_i for the sequent $\overline{P}; \exists_{\theta\diamond\eta} c_2 \vdash \overline{A_i}$ for each $i \in \{1, \ldots, i-1\} \cup \{i+1, \ldots, k\}$ and Π_* for the sequent $\overline{P}; \exists_{\theta_2\diamond\theta_1'} c_2 \vdash \overline{\mathbf{Tl}}$.

Consider the first backchain step in d, and note that the following is also a valid backchain step:

$$\mathbf{G} : \langle n, c_1\rangle \xrightarrow{\langle \theta_2\diamond\theta_1\diamond\eta, \theta_2{}^\sharp(\theta_1{}^\sharp \mathbf{G}_a \otimes \theta_1'{}^\sharp cl \otimes \theta_1{}^\sharp \mathbf{G}_b)\rangle} \rightsquigarrow \theta_2{}^\sharp \mathbf{G}' : \langle \rho, c_2\rangle \ .$$

Therefore, by Lemma A.15, there is a derived inference rule for $\overline{P}; \exists_{\theta\diamond\eta} c_2 \vdash \overline{\mathbf{G}}$ from the premises $\overline{P}; \exists_{\theta\diamond\eta} c_2 \vdash \overline{A_i}$ for $i \in \{1, \ldots, i-1\} \cup \{i+1, \ldots, k\}$ and $\overline{P}; \exists_{\theta_2\diamond\theta_1'} c_2 \vdash \overline{\mathbf{Tl}}$. By composing this inference rule with the proof trees Π_* and all the Π_i's, we obtain a proof of $\overline{P}; \exists_{\theta\diamond\eta} c_2 \vdash \overline{\mathbf{G}}$.

If the first step of d is obtained by an arrow reduction pair, we have:

$$d = \mathbf{G}\eta : \langle n, c_1\rangle \xrightarrow{\langle \theta_1, id_{\theta_1{}^\sharp \mathbf{G}}\rangle} \rightsquigarrow \mathbf{G}\eta\theta_1 : \langle m, c'\rangle \cdot d' \ ,$$

with $\mathsf{answer}(d') = \theta_2$ and $\theta_2 \diamond \theta_1 = \theta$. By inductive hypothesis, we have a proof of the sequent $\overline{P}; \exists_{\theta_2\diamond\theta_1\diamond\eta} \vdash \overline{\mathbf{G}}$, which is exactly what we were looking for. ∎