Encapsulating data in Logic Programming via Categorical Constraints

Technical Report*

James Lipton Robert McGrail
Dept. of Mathematics
Wesleyan University
http://www.cs.wesleyan.edu/

July 9, 1998

Abstract

We define a framework for writing executable declarative specifications which incorporate categorical constraints on data, Horn Clauses and datatype specification over finite-product categories. We construct a generic extension of a base syntactic category of constraints in which arrows correspond to resolution proofs subject to the specified data constraints.

1 Introduction

Much of the research in logic programming is aimed at expanding the expressive power and efficiency of declarative languages without compromising the logical transparency commitment: programs should (almost) read like specifications. One approach is to place more expressive power and more of the control components into the logic itself, possibly by expanding the scope of the underlying mathematical formalism. This has been the goal of constraint logic programming (CLP, Set constraints, Prolog III), and extensions to higher-order and linear logic, to name a few such efforts. This paper is a step in this direction. Rather than expand the logic itself, we consider two extensions of the underlying *syntactical* foundation, using fairly simple categorical tools. Categorical syntax and proof theory for logic programming permits a powerful extension of conventional logic programming to be built in a manner that does not compromise declarity and that admits a similar semantical treatment to conventional logic programming.

The encoding dilemma. The main contribution of the paper is a new approach to defining syntactic extensions to specify data types and constraints as independent components, somewhat in the spirit of modules in functional programming.

This information is processed by extending the logic programming interpreter dynamically with new rules. In traditional logic programming, this kind of information is coded directly into the Horn logic, often very cleverly, in a way that may obscure the intended meaning of the code, despite the fact that it is logical. Soundness and completeness of the target logic says nothing about the correctness of the encoding. We propose expanding the specification formalism in a way that

*some of these results appeared in proc. ALP98, LNCS, Springer.

admits direct definitions of data and constraint information that are already mathematically clear, or verified elsewhere using appropriate tools. We consider a datatype defintion construct and a very liberal notion of constraint based on finite product categories, of which equational constraint systems are a special case. We make use of a categorical foundation developed in [9], and similar in spirit to, e.g. [1, 4, 5, 21]. Although the framework admits extensions to Hereditarily Harrop or Higher-order logic, we will only consider Horn logic here.

An Example: It will be useful to give an example of the kind of code that is admitted by the extensions discussed in this paper.

```
begin module "list"
 datatype 'a list =  nil | cons of 'a * 'a list
  begin
    fun length nil = (0:int)
     |  length (cons (a,t)) = 1 + length t
  end;
 even(0:int).
 even(X) :- even(X - 2).

 length_of(Z, length Z).
end module;
? - list even (cons(2,(cons(4,(cons(6,nil)))))).
-  yes
? - pred[length](cons (2,(cons (4,(cons (5,nil))))),A).
-  A = 3
? - length_of((cons (2,(cons (4,(cons (5,nil)))))),A).
-  A = 3
```

It is assumed that a datatype `int`, together with operators `+,*,-` of type `int -> int`, constants `0,1,2,...:int` and equational constraints are defined elsewhere.

The predicate `pred[length]` is not specified by the user, but is built up automatically from the function definition for `length`, whereas (for no good reason other that to illustrate the difference) `length_of` is a predicate with the same meaning *hand coded* by the user[1] The query

```
?-length_of((cons (2,(cons (4,(cons (5,nil)))))),A+2).
```

yields A=1 by unification of `((cons (2,(cons (4,(cons (5,nil)))))),A+2)` with `(Z,length Z)` which is carried out in the appropriate category by taking the pullback of the two corresponding arrows, as described below. The predicate `list even` is also not defined directly by the user, only `even`.

In this paper we show how to define a category with arrows corresponding to resolution proofs with a generalized notion of unification with constraints such as:

$$\texttt{length\_of((cons (2,(cons (4,(cons (5,nil)))))),A+2)} \overset{A=1}{\rightsquigarrow} \Box.$$

and with new proof steps that incorporate data type information

$$\texttt{list } X \texttt{ (cons(a,t))} \rightsquigarrow X\texttt{(a),list } X \texttt{ (cons(a,t))}$$

which are treated as dynamic updates of the basic Horn clause interpreter.

What we will not discuss here are proof strategies: algorithms for searching the category of proofs or the category of non-deterministic resolutions defined in the paper, or for computing pullbacks. This is of course extremely relevant, and quite a harder matter to resolve than the subject matter of this paper. What we show here is that in principle a correct interpreter can be dynamically generated from constraint data expressed in a modular discipline that is both a specification at top level, and correctly executable code.

[1]The point of providing an automatically generated predicate `pred[length]` is that the function `length` could be made *private* to the data type. This is in keeping with the view that adding datatypes should not necessarily mean turning logic programming into a functional programming language.

Related Work: Module proposals, and data type definitions for logic programming have appeared since the early 1980's, notably [12, 2]. Our contribution here is to introduce a new technique for lifting certain constraints on data to predicates on that data and proofs between them, in the general framework of a categorically defined constraint domain.

Categorical approaches to logic programming features appeared in the mid 1980's in Rydeheard and Burstall's categorical treatment of unification [24] based on ideas of Burstall and Goguen. Since then research using categories to analyze or generalize different facets of declarative programming has been carried out by Corradini, Asperti and Montanari [1, 5, 4], Panangaden, Scott, Seely, Saraswat, [19], Power and Kinoshita [21] Diaconescu [8], Pym [22], Orejas, Ehrig and Pino [18], and Finkelstein, Freyd and Lipton [9, 10].

2 Logic Programming Categories

In the remainder of this paper, we build on the basic categorical framework defined in [9], based on standard approaches to categorical logic, e.g. [16, 11]. It is briefly sketched here.

Categorical Logic: In a nutshell, the categorical representation of the syntax of logic programs is as follows: an FP category (category with finite products) $\mathbb{C}$ is taken as a generalization of to the Herbrand Universe. It serves as a generalized program signature as well: it supplies the basic sorts (objects), function symbols of sort (σ, ρ) (arrows $\sigma \xrightarrow{f} \rho$), and constants of sort ρ (arrows of the form $\mathbf{1} \xrightarrow{a} \rho$, where $\mathbf{1}$ is the terminal object). Since it is closed under allowed compositions, it supplies the terms as well. Often, only a distinguished class of arrows (closed under composition) interest us as program terms, and are so identified. Predicates are subobjects of their sort, with instances given by pulling back along term arrows. In order to make the right kind of subobjects available, special categories $\mathbb{C}[X_1, \ldots, X_n]$ or $\mathbb{C}[\sigma]$ of generic predicates will be built on top of $\mathbb{C}$ subject to different constraints depending on the extension we are trying to formalize. In these categories, predicates are functors generated by spans emanating from their sort.

We can recover traditional logic programming over the Herbrand universe in this framework, if we pick $\mathbb{C}$ to be the so-called the Lawvere Algebraic Theory for a one-sorted signature with at least one constant (considered in e.g. [9]) and with no equations. Here, arrows correspond to (projections of vectors of) terms of the Herbrand Universe. We call this the *Herbrand Category* for the signature. The more general framework of arbitrary finite-product categories and generic predicate extensions allows us to *build-in* constraint information and data types into the syntax, as well as into the the structure of predicates. In such categories, unification of two arrows u and v with a common target (corresponding to two terms of the same sort which are *standardized apart*) amounts to finding a pair of substitutions (arrows) with a common source θ, ψ making a commuting square, i.e. such that $\psi v = \theta u$.

Most general unifiers yield pullback diagrams, when they exist. Note that separating the domains of terms u and v is what standardizes them apart[2]. In extensions of Horn logic, such as λ-prolog, queries and programs may share variables, which requires explicit sharing of domains. Unification then reduces to equalizing the arrows (see [10])

For basic concepts of category theory, we refer the reader to [11], and for the elements of indexed category theory, see [6]. We note that compositions of arrows are written in diagrammatic order: $A \xrightarrow{f} B$ composed with $B \xrightarrow{g} C$ is $A \xrightarrow{fg} C$.

[2]If the ambient category is the Herbrand Category, the occurs-check will be automatically enforced.

2.1 Generic Predicates and Unordered Goals

A predicate symbol R of type σ may be modelled in a category with finite products by a monic arrow with target σ. Instantiation (and hence, substitution) by terms of sort σ can be modelled by pullbacks along the appropriate arrow:

$$\begin{tikzcd} R(t) \arrow[r] \arrow[d, tail] \arrow[dr, phantom, "\lrcorner", very near start] & R \arrow[d, tail] \\ \alpha \arrow[r, "t"] & \sigma \end{tikzcd}$$

Any distinguished family of monics can play the role of predicates in a logic program, but if they are to make sense in logic program syntax such a collection should be closed under pullbacks[3] and they should not be chosen to conflict with the meaning of the logic program in which they will be used. Unlerss one wishes to constrain program predicates in advance (a question we will take up subsequently), the predicate A should not be true of any instance $A(t)$ in the syntactic category, until information about the program is somehow incorporated into the categorical structure. In short: they should behave like second-order predicate variables. We thus seek a notion of *generic* or *freely adjoined predicate*.

Definition 1 (Generic Predicates) *Let X be a subobject of some object b in a finite product category $\mathbb{C}$, and let $\mathcal{D}$ be a family of arrows in $\mathbb{C}$ targeted at b. We say X is a* **generic subobject** *of b with respect to the maps $\mathcal{D}$ if*

- *For every arrow t in $\mathcal{D}$ the pullback $t^{\#}(X)$ exists.*
- *No such pullback is an isomorphism.*

Definition 2 *Let $\mathbb{C}$ be an FP category and $\vec{b} = b_1, \ldots, b_n$ a sequence of objects of $\mathbb{C}$. Then $\mathbb{C}[X_1, \ldots, X_n]$, the* category obtained from $\mathbb{C}$ by freely adjoining indeterminate subobjects of $\vec{b}$ *is defined as follows:*

objects: *pairs $\langle A, S\rangle$ where $A \in |\mathbb{C}|$ and S is a sequence $S_1, \ldots, S_n$ of finite sets $S_i \subset Hom_{\mathbb{C}}(A, b_i)$,*

arrows: *are triples $\langle A, S\rangle \xrightarrow{f} \langle B, T\rangle$ where $A \xrightarrow{f} B$ is an arrow in $\mathbb{C}$ and $fT \subset S$, that is to say, for every i, $(1 \leq i \leq n)$ and every $t \in T_i$, $ft_i \in S_i$. The arrow f in $\mathbb{C}$ is called the* **label** *of $\langle A, S\rangle \xrightarrow{f} \langle B, T\rangle$. Composition of arrows is inherited from $\mathbb{C}$. Two arrows $\langle A, S\rangle \xrightarrow{f} \langle B, T\rangle$ and $\langle A', S'\rangle \xrightarrow{f'} \langle B', T'\rangle$ are* **equal** *if they have the same domain and range and if $f = f'$ in $\mathbb{C}$.*

Given an object $\langle A, S\rangle$ we will use the notation tS, where t is an arrow in $\mathbb{C}$ with target A, to mean the sequence $tS_1, \ldots, tS_n$ where $tS_i = \{ts : s \in S_i\}$. Notice that an arrow in $\mathbb{C}[X_1, \ldots, X_n]$, may have an identity arrow in $\mathbb{C}$ as a label, and not even be an isomorphism in $\mathbb{C}[X_1, \ldots, X_n]$. We will be paying special attention to a certain class of such arrows.

Theorem 3 *Let $\mathbb{C}$ be a finite product category. The category $\mathbb{C}[X_1, \ldots, X_n]$ has*

- *a terminal object $\langle \mathbf{1}, \vec{\emptyset}\rangle$, where $\vec{\emptyset}$ is the sequence $\emptyset, \ldots, \emptyset$ of length n,*
- *products: $\langle A, S\rangle \times \langle B, T\rangle = \langle A \times B, \pi_1 S \cup \pi_2 T\rangle$ where $A \xleftarrow{\pi_1} A \times B \xrightarrow{\pi_2} B$ is a product in $\mathbb{C}$.*

Furthermore, the functor $\mathbb{C} \xrightarrow{\iota} \mathbb{C}[X_1, \ldots, X_n]$ given by mapping objects A to $\langle A, \vec{\emptyset}\rangle$ and arrows $A \xrightarrow{f} B$ to $\langle A, \vec{\emptyset}\rangle \xrightarrow{f} \langle B, \vec{\emptyset}\rangle$, is a limit-preserving, full embedding.

[3] If the predicates are not stable under pullbacks of arrows designated as program terms, then certain instances may not even exist in the syntax, which is a rather unusual form of failure for a logic program query.

Limit preservation follows from the fact that ι has a left adjoint, namely the forgetful functor U taking objects $\langle A, S\rangle$ to A and arrows to their labels.

Lemma 4 *Addition of indeterminate subobjects simultaneously, sequentially, or in permuted order results in isomorphic categories. More precisely:*

1. $\mathbb{C}[X_1, \ldots, X_n] \simeq \mathbb{C}[X_1] \cdots [X_n]$.
2. *Let σ be a permutation of the first n positive integers. Then*

$$\mathbb{C}[X_1, \ldots, X_n] \simeq \mathbb{C}[X_{\sigma(1)}, \ldots, X_{\sigma(n)}]$$

.

Proof. Straightforward. □

Definition 5 *In $\mathbb{C}[X_1, \ldots, X_n]$ define the* indeterminate subobjects $X_1, \ldots, X_n$ *of sorts $b_1, \ldots, b_n$ respectively, to be the subobjects $\langle b_i, J^i\rangle \overset{id_{b_i}}{\rightarrowtail} \langle b, \vec{\emptyset}\rangle$, where the J^i are the* "basis vectors"

$$(J^i)_k = \begin{cases} \emptyset & \text{if} \quad i \neq k \\ \{id_{b_i}\} & o.w. \end{cases}$$

Theorem 6 *The indeterminate subobjects X_i of b_i are generic with respect to the maps (targeted at $\langle b_i, \emptyset\rangle$) in the image of $\mathbb{C} \xrightarrow{\iota} \mathbb{C}[X_1, \ldots, X_n]$.*

Proof. The following diagram is a pullback for any arrow $\langle A, \vec{\emptyset}\rangle \xrightarrow{t} \langle b_i, \vec{\emptyset}\rangle$:

$$\begin{tikzcd} \langle A, tJ^i\rangle \arrow[r, "t"] \arrow[d, tail, "id_A"'] \arrow[dr, phantom, "\lrcorner", very near start] & \langle b_i, \{id\}\rangle \arrow[d, tail, "id_{b_i}"] \\ \langle A, \vec{\emptyset}\rangle \arrow[r, "t"'] & \langle b_i, \vec{\emptyset}\rangle \end{tikzcd}$$

so $X(t) = \langle A, tJ^i\rangle \overset{id_A}{\rightarrowtail} \langle A, \vec{\emptyset}\rangle$ exists for all appropriate t. This arrow cannot be an isomorphism in $\mathbb{C}[X_1, \ldots, X_n]$: its inverse, which would have to be labelled with id_A, would have to satisfy $id_A t \in \emptyset$. □

Objects of the form $X(t) = \langle A, tJ^i\rangle$ will be called **atomic** predicates. If A is an object of $\mathbb{C}$, we say that the monic $\langle B, S\rangle \overset{f}{\rightarrowtail} \langle A, \vec{\emptyset}\rangle$ is a canonical (representative of a) subobject of $\langle A, \vec{\emptyset}\rangle$ if B is A and the monic f is id_A. Observe that every object $\langle A, S\rangle$ of $\mathbb{C}[X_1, \ldots, X_n]$ is a canonical subobject of "its sort" $\langle A, \vec{\emptyset}\rangle$. This allows us to define a natural indexed structure [6] for $\mathbb{C}[X_1, \ldots, X_n]$ over $\mathbb{C}$.

Definition 7 *For each object A of $\mathbb{C}$, let $\mathbb{C}_A[X_1, \ldots, X_n]$ be the category whose objects are arrows in $\mathbb{C}[X_1, \ldots, X_n]$ of the form $\langle A, S\rangle \xrightarrow{id_A} \langle A, \vec{\emptyset}\rangle$, and with morphisms given by arrows between their sources labelled by the identity on A in $\mathbb{C}$. Then let $p : \mathbb{C} \longrightarrow \mathbb{CAT}$ be the strict indexed category given by*

- $p(A) = \mathbb{C}_A[X_1, \ldots, X_n]$

- $p(A \xrightarrow{f} B) = f^{\#} : \mathbb{C}_B[X_1, \ldots, X_n] \longrightarrow \mathbb{C}_A[X_1, \ldots, X_n]$

Notice that the pullback operation referred to by $f^{\#}$ maps $\langle B, S\rangle \xrightarrow{id_B} \langle B, \vec{\emptyset}\rangle$ to $\langle A, fS\rangle \xrightarrow{id_A} \langle A, \vec{\emptyset}\rangle$. Thus $(fg)^{\#}$ is precisely $g^{\#}f^{\#}$ on the nose. Now we define a "canonical intersection" functor for the indexed category p:

Definition 8 *Let $\cap : p \times p \longrightarrow p$ be defined by $\langle A, S\rangle \cap_A \langle A, T\rangle = \langle A, S \cup T\rangle$.*

By the remarks immediately preceding the definition, it is immediate that $\cap$ commutes with pullback, i.e. is a natural transformation.

The following theorems make precise the fact that $\mathbb{C}\,[X_1, \ldots, X_n]$ is called the category obtained by freely adjoining the indeterminate subobjects of the sorts $b_1, \ldots, b_n$.

Lemma 9 *Every object $\langle A, S\rangle$ is representable as (i.e. equal on the nose to) the canonical intersection*

$$\bigcap\{t^{\#}(X_i) : t \in S_i, 1 \le i \le n\}$$

where the pullbacks are canonical: $t^{\#}(X_i) = \langle A, tJ^i\rangle = \langle A, \emptyset \cdots \emptyset \underbrace{\{t\}}_{i} \emptyset \cdots \emptyset\rangle$.

Proof. Immediate: Since $S = \bigcup\{\{t\} : t \in S\}$, the indicated canonical intersection is precisely $\langle A, S\rangle$. $\Box$

Theorem 10 (Universal Mapping Property) *Suppose $F : \mathbb{C} \longrightarrow \mathbb{D}$ is a limit preserving functor from the finite-product category $\mathbb{C}$ to the finitely complete category $\mathbb{D}$, and that $F(b_i) = d_i$ for $1 \le i \le n$. Furthermore, let $B_1, \ldots, B_n$ be a sequence of subobjects of $d_1, \ldots, d_n$ respectively, in $\mathbb{D}$. Then there is a limit-preserving functor $F_{\vec{B}} : \mathbb{C}\,[X_1, \ldots, X_n] \longrightarrow \mathbb{D}$, unique up to isomorphism, such that the following diagram commutes.*

$$\begin{tikzcd} & \mathbb{C}\,[X_1, \ldots, X_n] \arrow[dr, "F_{\vec{B}}"] & \\ \mathbb{C} \arrow[ur, "\iota"] \arrow[rr, "F"'] & & \mathbb{D} \end{tikzcd}$$

$F_{\vec{B}}$ is called the **evaluation** *functor induced by the B_i.*

Proof. Define $F_{\vec{B}}$ on objects by $F_{\vec{B}}(\langle A, S\rangle) = \varprojlim\{F(t)^{\#}(B_i) : t \in S_i, 1 \le i \le n\}$. The universal mapping property of limits gives us the action on arrows: if $\langle A, S\rangle \xrightarrow{f} \langle A', S'\rangle$ is an arrow in $\mathbb{C}\,[X_1, \ldots, X_n]$ then $F_{\vec{B}}(\langle A, S\rangle)$, the limit of the family of monics $\{F(t)^{\#}(B_i) : t \in S_i, 1 \le i \le n\}$ targeted at FA, is also, by composing with $F(A \xrightarrow{f} B)$ and using properties of pullbacks and of arrows in $\mathbb{C}\,[X_1, \ldots, X_n]$, a cone over the family of monics $\{F(t)^{\#}(B_i) : t \in S'_i, 1 \le i \le n\}$. There is therefore a unique induced arrow $F\langle A, S\rangle \xrightarrow{\theta} F\langle A', S'\rangle$ which is the value of $F(\langle A, S\rangle \xrightarrow{f} \langle A', S'\rangle)$. The details, and those of the proof of limit preservation, are left to the perseverant reader. $\Box$

We are interested in a category $\mathbb{D}$ with richer structure, in which case we are able to sharpen this result a bit.

Corollary 11 *Assume the category $\mathbb{D}$ in the preceding theorem is $\mathsf{Set}^{\mathbb{C}^o}$ and that F is the Yoneda embedding. Choose the sequence of subobjects B_i of $Fb_i = Hom_{\mathbb{C}}(\_, b_i)$ to be* canonical, *that is to say, pointwise subsets of Fb_i, and take limits in $\mathsf{Set}^{\mathbb{C}^o}$ to be given pointwise (not just up to isomorphism, but on the nose). Then the evaluation functor $F_{\vec{B}}$ of the preceding theorem is unique.*

2.1.1 Unordered Goals

The most elementary notion of query, or of state in a logic program, is that of a conjunction of atoms

$$X_{i_1}(t_1), \ldots, X_{i_n}(t_n) \tag{1}$$

where the X_{i_j} are program predicate symbols. We call these *basic* goals.

A first approximation to syntactic goals is already present in the category $\mathbb{C}[X_1, \ldots, X_n]$ whose objects are effectively *unordered goals*. By the representation lemma (9) above, every object $\langle A, S\rangle$ in $\mathbb{C}[X_1, \ldots, X_n]$ is an intersection

$$\bigcap\{t^{\#}(X_i) : t \in S_i, 1 \leq i \leq n\}. \tag{2}$$

and can be thought of as a non-deterministic image of the corresponding ordered goal (1) above.

These intersections are free in the sense that one can recover all components $t^{\#}(X_i)$ from them, i.e. by reading off the arrows in the S_i, and in the sense of theorem 10. Since the S_i are sets, they cannot capture order or repititions of atoms within goals. An ordered counterpart will be defined below.

Definition 12 *Let $\mathbb{C}$ be a finite product category, and $X_1, \ldots, X_n$ a sequence of generic predicates over $\mathbb{C}$ of sorts $b_1, \ldots, b_n$. An* **interpretation** *is an evaluation functor extending the Yoneda embedding, assigning to each X_i some canonical subobject B_i of $Hom_{\mathbb{C}}(\_, b_i)$ as in corollary 11.*

In other words, an interpretation is a functor

$$[\![\,]\!] : \mathbb{C}[X_1, \ldots, X_n] \longrightarrow \mathsf{Set}^{\mathbb{C}^o}$$

- agreeing with the Yoneda embedding on $\mathbb{C}$, and
- mapping $\langle A, S\rangle$ to $\bigcap\{([\![t]\!])^{\#}(B_i) : t \in S_i, 1 \leq i \leq n\}$,

where $[\![t]\!]$ means the Yoneda image of t.

It is easy to check that interpretations form a complete lattice under the pointwise order.

2.2 Two Proof-theoretic Categories

Fix a sequence of sorts $\sigma = \sigma_1, \ldots, \sigma_n$, $\sigma_i \in |\mathbb{C}|$ and a family $X_1, \ldots, X_n$ of generic predicates $X_1, \ldots, X_n$ with X_i of sort σ_i, as in the construction of the categories in the preceding section. We will refer to objects $X_i(t_i) = t_i J^i$ of sort A, where $A \xrightarrow[t]{} \sigma_i$ is an arrow in $\mathbb{C}$, as atomic goals or predicates (of sort A), objects $\langle A, S\rangle$ of $\mathbb{C}[X_1, \ldots, X_n]$ as unordered (or non-deterministic) goals (of sort A), and sequences $X_{i_1}(t_1), \ldots, X_{i_m}(t_m)$, where each atomic predicate $X_{i_j}(t_j)$ is of sort A, as ordered goals of sort A.

Note that there is a "forgetful" function β which takes ordered goals to the underlying unordered ones:

$$\beta(X_{i_1}(t_1), \ldots, X_{i_m}(t_m)) = \langle A, S\rangle, \tag{3}$$

where $S_k = \{t_j : i_j = k\}$ that is to say, the set of all terms occurring as arguments to the kth generic predicate, wherever it occurs (0 or more times) in $X_{i_1}(t_1), \ldots, X_{i_m}(t_m)$. Thus $\langle A, S\rangle = \bigcap\{t^{\#}(X_i) : t \in S_i, 1 \leq i \leq n\}$ as discussed in the previous section.

It will be convenient below to describe the following operation on ordered and unordered goals, both given the same name del^k_t, which removes one occurrence (the first one, in the ordered case) of the atomic goal $X_k(t)$ if it exists, and returns the original goal unchanged, otherwise. More formally, in the unordered case: $\mathsf{del}^k_t\langle A, S\rangle = \langle A, T\rangle$ where $T_i = S_i$ if $i \neq k$ and $T_k = S_k \setminus \{t\}$.

Definition 13 *A Horn Program over $\mathbb{C}$ (in the predicates $X_1, \ldots, X_n$) is a finite set of triples $\langle A, G, X(t)\rangle$ where G is an ordered goal, and $X(t)$ an atomic goal, both of sort A. An unordered Horn program is a similar set of triples $\langle A, G, X(t)\rangle$, but where G is unordered. The triples are called (ordered or unordered) clauses, and may be written $G \Rightarrow X(t)$ when the sort is understood from context.*

Note that the "forgetful" function β in (3) extends naturally to a map from ordered clauses to unordered ones.

Definition 14 *Let P be a Horn program. A P-SLD proof step between ordered goals G_1 and G_2, of sorts A_1 and A_2, respectively, with substitution θ, and program clause $c = \langle A, H, X_k(u)\rangle$, denoted*

$$G_1 \overset{\theta,c}{\rightsquigarrow} G_2$$

is a 4-ary relation (relating G_1, θ, c, G_2) defined by:

- $A_2 \xrightarrow{\theta} A_1$ *is an arrow in $\mathbb{C}$, and*
- *there is an arrow $A_1 \xrightarrow[t]{} \sigma_k$ in $\mathbb{C}$, such that $X_k(t)$ is an atomic subgoal of G_1, that is to say, $G_1 = G, X_k(t), G'$,*
- *there is an arrow $A_2 rTo^{\psi} A$ such that $\theta t = \psi u$ and*
- $G_2 = \theta G, \psi H, \theta G'$.

If $\langle A_1, S\rangle, \langle A_2, T\rangle$ are unordered goals, an sld-step $\langle A_1, S\rangle \overset{\theta,c}{\rightsquigarrow} \langle A_2, T\rangle$ between them exists when a clause c and unifying arrows θ and ψ exist, as above, and $T = \theta(\mathsf{del}_t^k S) \cap \psi H$. In both the ordered and unordered cases, the arrow θ is called the substitution of the SLD step, and c its clause.

We define an SLD-*sequence* between goals, $G \rightsquigarrow \overset{\theta}{\cdots} \rightsquigarrow G'$ to be the ternary relation that obtains when there is a sequence of goals $G = G_0, \ldots, G_n = G'$ and SLD-steps between each G_{i-1} and G_i with substitution θ_i, and θ is the composition $\theta_n \cdots \theta_1$. The "reflexive case"$n = 0$ and $\theta = id$ is allowed.

Definition 15 *The category $\mathbb{C}_P^{SLD}$ of unordered SLD proofs over program P has the objects of $\mathbb{C}[X_1, \ldots, X_n]$ as its objects, and arrows $\langle A, S\rangle \xrightarrow{\theta} \langle B, T\rangle$ where $\langle B, T\rangle \rightsquigarrow \overset{\theta}{\cdots} \rightsquigarrow \langle A, S\rangle$ is an SLD sequence. We call $A \xrightarrow{\theta} B$ in $\mathbb{C}$ the* label *of this arrow of $\mathbb{C}_P^{SLD}$.*

2.2.1 The Operational Category $\mathbb{C}_{\mathsf{P}}$

We now modify the generic predicate construction $\mathbb{C}[X_1, \ldots, X_n]$ to produce a category of predicates $\mathbb{C}_{\mathsf{P}}$which are generic *modulo the clauses in program* P. $\mathbb{C}_{\mathsf{P}}$ is a categorical (and constraint-sensitive) counterpart to the operational C-semantics of Levi et. al. [15], in the sense that objects are goals modulo operational equivalence, and arrows are open substitutions between them. Later we will study how datatype information acts on both $\mathbb{C}_{\mathsf{P}}$ and $\mathbb{C}_P^{SLD}$ to produce dynamically updated predicates, operational semantics and proofs.

Definition 16 *Let A be an object in $\mathbb{C}$, and T a monotone operator on sets of arrows with a common source in $\mathbb{C}$. Then a family S of arrows with source A is said to be closed under T if $T(S) = S$. S is T-***generated** *by a family of arrows emanating from A if $S = T(U)$. We will also write $\langle U\rangle$ for $T(U)$ when the closure operator is understood from context.*

Definition 17 *Let $\mathbb{C}$ be a finite-product category, and P a Horn clause program over $\mathbb{C}[X_1, \ldots, X_n]$. The category $\mathbb{C}_{\mathsf{P}}$ is given by the following data:*

objects: *Pairs (A, S) where A is an object of $\mathbb{C}$, and S is a sequence $S_1, \ldots, S_n$, where each S_i is a set of arrows from A to σ_i, closed* with respect to the following **clausal** closure condition:
For each clause of sort α in P

$$X_{i_1}(t_1), \ldots, X_{i_n}(t_n) \Rightarrow X(t) \tag{4}$$

where X, X_{i_k} are generic predicates of sort σ_j, σ_{i_k}, $\alpha \xrightarrow{t} \sigma$ and $\alpha \xrightarrow{t_k} \sigma_{i_k}$ arrows in $\mathbb{C}$, and for each arrow $A \xrightarrow{\varphi} \alpha$ satisfying

$$\varphi t_1 \in S_{i_1}, \ldots, \varphi t_n \in S_{i_n} \tag{5}$$

we must have $\varphi t \in S_j$.

arrows *are triples $(A, S) \xrightarrow{f} (B, V)$ such that $A \xrightarrow{f} B$ (the* label*) is an arrow in $\mathbb{C}$, and $fV \subset S$. Composition is defined by composing labels.*

If we let T^A_{P} be the operator on sets of arrows with a common source A given by

$$T^A_{\mathsf{P}}(S) = \bigcup_{c \in \mathsf{P}} \{\varphi t : A \xrightarrow{\varphi} \alpha \text{ and } \varphi t_1 \ \ldots, \varphi t_n \in S\}$$

then the requirement on S in the preceding definition is equivalent to being T^A_{P}-generated.

Lemma 18 *There is a faithful functor $\iota_{\mathsf{P}} : \mathbb{C}[X_1, \ldots, X_n] \longrightarrow \mathbb{C}_{\mathsf{P}}$ given by $(A, U) \mapsto (A, \langle U \rangle)$ on objects, and which preserves labels of arrows.*

Proof. Since the closure operator T^A_{P} is monotone, ι_{P} maps arrows to arrows. Fidelity and functoriality is immediate. $\square$

2.3 Semantics

Let P be a program, $\mathcal{G}^{\mathsf{P}}$ the collection of ordered goals over P, and $[\![\ ]\!]$ an interpretation into $\mathsf{Set}^{\mathbb{C}^o}$ as in definition (12), and β the forgetful map from ordered to unordered goals. Then observe that $\beta[\![\ ]\!]$ is a function $\mathcal{G}^{\mathsf{P}} \longrightarrow \mathsf{Set}^{\mathbb{C}^o}$. We sometimes overload the symbol $[\![\ ]\!]$ to denote $\beta[\![\ ]\!]$, since context will always make clear what is meant. We call the composition $\beta\iota_{\mathsf{P}} : \mathcal{G}^{\mathsf{P}} \longrightarrow \mathbb{C}_{\mathsf{P}}$ the representation of goals in the category $\mathbb{C}_{\mathsf{P}}$.

Note that by corollary (11), *the value of $[\![\ ]\!]$ on goals is completely determined by its value on generic predicates.*

We now give a quick sketch of some results on semantics from [9, 10]. We give a categorical analogue of the Kowalski-van Emden bottom-up semantics. Proofs can be founded in the cited references, and are, for the most part, omitted.

Definition 19 *An interpretation $[\![\ ]\!]$ is a model of program P if for every clause $tl_{cl} \Rightarrow hd_{cl}$ we have $[\![tl]\!] \subset [\![hd]\!]$. A goal G of sort α is said to be* **true** *in the interpretation if the image $[\![\beta(G)]\!] \in \mathsf{Set}^{\mathbb{C}^o}$ of the monic*

$$\beta(G) \xrightarrow{id_\alpha} (\alpha, \emptyset)$$

is an isomorphism.

In the following discussion we will use the notation $cl \in \mathsf{P}$ to refer to the fact that cl is an ordered clause in the program P. We also refer to the empty goal of sort σ by $\Box_\sigma$. The image of this goal in $\mathbb{C}[X_1, \ldots, X_n]$ under β is $(\sigma, \emptyset)$, which is the entire subobject $(\sigma, \emptyset) = (\sigma, \emptyset)$. Since functors preserve identity arrows, it must be mapped by any interpretation to the identity $\mathbb{C}(\_, \sigma) = \mathbb{C}(\_, \sigma)$.

Lemma 20 (Soundness) *Let $[\![\ ]\!]$ be a model of program P, and G_1 and G_2 goals of sorts α_1, α_2 respectively, and $G_1 \rightsquigarrow \overset{\theta,c}{\cdots} \rightsquigarrow G_2$ a resolution proof with answer substitution $\alpha_2 \xrightarrow{\theta} \alpha_1$. Then $[\![G_2]\!] \subset [\![\theta G_1]\!]$. In particular, if G_2 is $\Box_{\alpha_2}$ then θG_1 is true in the model.*

We now define a categorical analogue of the T_P operator of Kowalski and Van Emden, an operator E_P on the lattice of interpretations.

Definition 21 *Let $[\![\ ]\!]$ be an interpretation and $X_1, \ldots, X_n$ the sequence of generic predicates in program P. Then*

$$E_P([\![\,]\!])(X_i) = \bigcup_{tl \Rightarrow X_i(t) \in P} \mathsf{Im}_{[\![\mathsf{t}]\!]}\,([\![\mathsf{tl}_{\mathsf{cl}}]\!])$$

where, for each t occurring in the head of a clause in P of the form $tl \Rightarrow X_i(t)$, $\mathsf{Im}_{[\![\mathsf{t}]\!]}$ denotes the image along the arrow $[\![t]\!]$ in $\mathsf{Set}^{\mathbb{C}^o}$.

In [9, 10] it is shown that E_P is a continuous operator on the lattice of interpretations, with a least fixed point $[\![\ ]\!]^*$ (called the Herbrand interpretation for P).

Lemma 22 *An interpretation $[\![\ ]\!]$ is a model of program P if and only if it is a pre-fixed point of E_P (that is to say, $E_P([\![\ ]\!]) \subseteq [\![\ ]\!]$) and hence, if and only if $[\![\ ]\!]^* \subseteq [\![\ ]\!]$.*

The following is established in [9, 10]

Theorem 23 (Completeness) *If P is a program, $[\![\ ]\!]^*$ its Herbrand interpretation and G a goal, $[\![G]\!]^*$ is an isomorphism if and only if there is an SLD proof $G \rightsquigarrow \overset{id}{\cdots} \rightsquigarrow \Box$*

We now establish some properties of the category $\mathbb{C}_{\mathsf{P}}$. Since it is a finite-product category, arrows in $\mathbb{C}_{\mathsf{P}}$ are not, strictly speaking, SLD proof steps (in reverse), but a bit more. Arrows in $\mathbb{C}_{\mathsf{P}}$ correspond to *extended* resolution steps that include weakening of goals and instantiation of goals, as well as products of basic SLD sequences that can be thought of as parallel resolutions of multiple goals. But the present category suffices for our purposes, since the only *global* proofs $\Box_\alpha \xrightarrow{\theta} \beta\iota_{\mathsf{P}}(G)$ correspond to real SLD proofs $G \rightsquigarrow \overset{\theta}{\cdots} \rightsquigarrow \Box_\alpha$.

Theorem 24 (Soundness of $\beta \circ \iota_{\mathsf{P}}$) *Let $G_1 \rightsquigarrow \overset{\theta}{\cdots} \rightsquigarrow G_2$ be an SLD sequence, and let $(\alpha_1, \langle U_1 \rangle)$ and $(\alpha_2, \langle U_2 \rangle)$ be the images in $\mathbb{C}_{\mathsf{P}}$ of G_1 and G_2 under $\beta \circ \iota_{\mathsf{P}}$. Then $G_2 \xrightarrow{\theta} G_1$ is an arrow in $\mathbb{C}_{\mathsf{P}}$.*

Proof. Observe that the closure condition (5) of definition (17) guarantees that for each clause $tl \Rightarrow hd$ of P, the arrow $(\alpha, \langle S_{tl} \rangle) \xrightarrow{id_\alpha} (\alpha, \langle S_{hd} \rangle)$ is in $\mathbb{C}_{\mathsf{P}}$, for $\beta\iota_{\mathsf{P}}(hd) = (\alpha, \langle S_{hd} \rangle)$ and $\beta\iota_{\mathsf{P}}(tl) = (\alpha, \langle S_{tl} \rangle)$, the requirement for arrows in $\mathbb{C}_{\mathsf{P}}$ being $S_{tl} \subseteq S_{hd}$ which is precisely the closure condition (5). This proves the claim for the trivial one-step resolution from the head of a clause to its tail. Using the fact that for any arrow θ targeted at α $(\alpha, \theta\langle S_{tl} \rangle) \xrightarrow{id_\alpha} (\alpha, \theta\langle S_{hd} \rangle)$ is also in $\mathbb{C}_{\mathsf{P}}$, and that $G_1 \rightsquigarrow \overset{\theta}{\cdots} \rightsquigarrow G_2$ implies $\theta G_1 \rightsquigarrow \overset{id}{\cdots} \rightsquigarrow G_2$, it is straightforward to show that any resolution sequence with answer substitution θ maps to an arrow with label θ in the opposite direction. $\Box$

Theorem 25 (Completeness of $\beta \circ \iota_{\mathsf{P}}$) *If $\Box_\alpha \xrightarrow{\theta} \beta\iota_{\mathsf{P}}G$ is an arrow in $\mathbb{C}_{\mathsf{P}}$, then there is an SLD proof $G \rightsquigarrow \overset{\theta}{\cdots} \rightsquigarrow \Box_\alpha$.*

Proof. The empty goal of sort α is represented by the object $(\alpha, \langle\emptyset\rangle)$ in $\mathbb{C}_{\mathsf{P}}$. Suppose $\beta\iota_{\mathsf{P}}G = (\gamma, \langle U\rangle)$. That is to say G is of the form $X_{i_1}(t_1), \ldots, X_{i_n}(t_n)$ and U is

$$\bigcap\{t^{\#}(X_i) : t \in U_i, 1 \leq i \leq n\}. \tag{6}$$

Then if $\Box_\alpha \xrightarrow{\theta} \beta\iota_{\mathsf{P}}G$ is an arrow in $\mathbb{C}_{\mathsf{P}}$, $\theta\langle U\rangle \subseteq \langle\emptyset\rangle$, that is to say, by the remarks about T_{P} following definition (17), every $X_{i_j}(t_j)$ is true in the Herbrand interpretation $[\![\ ]\!]^*$ of P for $(1 \leq j \leq n)$. By completeness for $[\![\ ]\!]^*$ (theorem 23) there is an SLD proof of θG from P, hence a proof of G with computed answer substitution θ. $\Box$

3 The Categories $\mathbb{C}[\sigma]$, $\mathbb{C}_{\mathsf{P}}[\sigma]$ and $\mathbb{C}_P^{SLD}[\sigma]$

Next, we generalize definition 2 so that the resultant construction will allow lifting of functorial datatypes and certain diagrams from the base category to the category of predicates. For the duration of this section $\mathbb{C}$ is an FP category, $\mathbb{D}$ is a complete category, $\mathbb{J}$ is a category, and $\mathcal{D}$ is a collection of product diagrams in $\mathbb{J}$, which we will call distinguished products, or **$\mathcal{D}$-products**.

Definition 26 ($\mathcal{D}$-Product Preservation) *We will say that the functor $H : J \longrightarrow \mathbb{B}$ preserves $\mathcal{D}$-products if H sends each $\mathcal{D}$-product to a product diagram in $\mathbb{B}$.*

For the rest of this paper $\sigma : \mathbb{J} \longrightarrow \mathbb{C}$ will be a functor that preserves $\mathcal{D}$-products and σ_i will denote the $\mathbb{C}$-object $\sigma(i)$.

Definition 27 ($\mathcal{D}$-Closure) *Let $A \in |\mathbb{C}|$, and $\mathcal{D}$ be a set of product diagrams in $\mathbb{C}$. A subfunctor S of $\mathbb{C}(A, \sigma(-))$ is $\mathcal{D}$-closed if it preserves $\mathcal{D}$-products. In other words, for every $\mathcal{D}$-product diagram for $i_1 \times \ldots \times i_n$, if there are arrows $f_k \in S(i_k)$ for each $k = 1, \ldots, n$, then $\langle f_1, \ldots, f_n\rangle \in S(i_1 \times \ldots \times i_n)$.*

In particular, if $\mathbf{1}_{\mathbb{J}} \in \mathcal{D}$, then $\sigma_{\mathbf{1}_{\mathbb{J}}} = \mathbf{1}_{\mathbb{C}}$ and $S(\mathbf{1}_{\mathbb{J}}) = \{!_A\}$.

Definition 28 ($\mathcal{D}$-Generating Sets) *Let Y be a collection of arrows emanating from A, each with target in the range of σ. Y is said to $\mathcal{D}$-generate S if S is the smallest $\mathcal{D}$-closed subfunctor of $\mathbb{C}(A, \sigma(-))$ containing all the arrows of Y. In that case we write $\ll Y \gg$ for S, and call it the $\mathcal{D}$-closure of Y.*

Definition 29 *Let $\mathbb{C}[\sigma]$ be defined as follows.*

objects: *pairs $\langle A, S\rangle$ where $A \in |\mathbb{C}|$ and S is a $\mathcal{D}$-closed subfunctor of $\mathbb{C}(A, \sigma(-))$.*

arrows: *are triples $\langle A, S\rangle \xrightarrow{h} \langle B, T\rangle$ where $A \xrightarrow{h} B$ is an arrow in $\mathbb{C}$ and $hT \subset S$, i.e. for any $j \in |\mathbb{J}|$ and $f \in T(j)$, $hf \in S(j)$. Again, we define a* **label** *as before, and compostion is inherited from $\mathbb{C}$.*

Observe that the functors S are no longer finitely generated.

A routine verification shows that if $\mathbb{J}$ is the discrete category on the set $\{1, \ldots, n\}$ and $\mathcal{D} = \emptyset$ then $\mathbb{C}[\sigma] = \mathbb{C}[X_1, \ldots, X_n]$. Hence Definition 2 is a special case of Definition 29.

Proposition 30 *If $\mathbb{C}$ is an FP category [resp. cartesian] then $\mathbb{C}[\sigma]$ is an FP category [resp. cartesian] and the functor $\mathbb{C} \xrightarrow{\iota} \mathbb{C}[\sigma]$ given by mapping objects A to $\langle A, \ll \emptyset \gg\rangle$, and arrows $A \xrightarrow{f} B$ to $\langle A, \ll \emptyset \gg\rangle \xrightarrow{f} \langle B, \ll \emptyset \gg\rangle$, is a full and faithful limit-preserving embedding.*

The proof is straightforward, but lengthy. The reader is referred to [17] for details.

Definition 31 *Let $\mathbb{C}(\sigma)$ stand for the full subcategory of $\mathbb{C}[\sigma]$ whose objects are*

$$|\iota(\mathbb{C})| \bigcup \{\langle \sigma_j, \ll Id_{\sigma_j} \gg\rangle | j \in |\mathbb{J}|\}.$$

Also, let $X : \mathbb{J} \longrightarrow \mathbb{C}(\sigma)$ be the functor taking objects $j \in |\mathbb{J}|$ to the pair $\langle \sigma_j, \ll Id_{\sigma_j} \gg\rangle$ and arrows $i \xrightarrow{h} j$ to $\langle \sigma_i, \ll Id_{\sigma_i} \gg\rangle \xrightarrow{\sigma_h} \langle \sigma_j, \ll Id_{\sigma_j} \gg\rangle$.

Finally, let $m : X \longrightarrow \iota \circ \sigma$ be the natural transformation defined by $m_j \equiv X_j \xrightarrow{id_{\sigma_j}} \iota(\sigma_j)$ where $j \in |\mathbb{J}|$.

It is straightforward to show the composite functor $\mathbb{J} \xrightarrow{X} \mathbb{C}(\sigma) \hookrightarrow \mathbb{C}[\sigma]$ preserves $\mathcal{D}$-products.

The monic m_j (and by abuse of language, its source X_j) will be called the **generic predicate of sort** σ_j since the family m exhibits similar characteristics to the generic predicates of Definition 1. In particular, for every $j \in |\mathbb{J}|$, $t \in \mathbb{C}\,(A, \sigma_j)$ the diagram

$$\begin{tikzcd}
\langle A, \ll t \gg\rangle \arrow[r, "t"] \arrow[d, "Id_A"'] \arrow[dr, phantom, "\lrcorner", very near start] & X_j \arrow[d, hook, "m_j"] \\
\langle A, \ll \emptyset \gg\rangle \arrow[r, "t"'] & \iota\sigma_j
\end{tikzcd}$$

is a pullback in $\mathbb{C}[\sigma]$. However, it may be the case that some of these pullbacks are indeed isomorphisms: the constraint information contained in σ may make some generic predicates true. For example, suppose $\mathbf{1}_{\mathbb{J}}$ is in $\mathcal{D}$. Then it follows that $X_{\mathbf{1}_{\mathbb{J}}} = \sigma_{\mathbf{1}_{\mathbb{J}}} = \mathbf{1}_{\mathbb{C}}$ so that $X_{\mathbf{1}_{\mathbb{J}}} \xrightarrow{m_{\mathbf{1}_{\mathbb{J}}}} \sigma_{\mathbf{1}_{\mathbb{J}}}$ is iso.

The "genericity" of the family m is made precise in the following theorem.

Theorem 32 (Universal Mapping Property II) *Let $\mathbb{D}$ be a complete category and suppose that $F : \mathbb{C}(\sigma) \longrightarrow \mathbb{D}$ is a functor such that*

- *$F \circ \iota : \mathbb{C} \longrightarrow \mathbb{D}$ is a cartesian representation;*
- *$F \circ X : \mathbb{J} \longrightarrow \mathbb{D}$ preserves $\mathcal{D}$-products; and*
- *$F(m) : F \circ X \longrightarrow F \circ \iota(\sigma)$ is a monic natural transformation.*

Then there exists a cartesian functor $\bar{F} : \mathbb{C}[\sigma] \longrightarrow \mathbb{D}$ unique up to isomorphism making the triangle

$$\begin{tikzcd}
& \mathbb{C}[\sigma] \arrow[dr, "\bar{F}"] & \\
\mathbb{C}(\sigma) \arrow[ur, hook] \arrow[rr, "F"'] & & \mathbb{D}
\end{tikzcd}$$

commute.

Proof(sketch). For $A \xrightarrow{t} \sigma_j$, let $\tilde{F}(t)$ be the pullback of $X_j \xrightarrow{Fm_j} F\iota\sigma_j$ along $F\iota(A) \xrightarrow[F(t)]{} F\iota\sigma_j$.

Define $\bar{F}\langle A, S\rangle$ on objects by $\bar{F}\langle A, S\rangle = \bigcap\{\tilde{F}(t) | t \in S(j), j \in |\mathbb{J}|\}$. $\bar{F}$ on arrows is just a consequence of the limit definition of $\bar{F}$ on objects. It follows that $\bar{F}$ has the stated properties. We refer the reader to [17] for the details. □

Corollary 33 *Assume the category $\mathbb{D}$ in Theorem 32 is $\mathsf{Set}^{\mathbb{C}^o}$ and that $F \circ \iota$ is the Yoneda embedding. Also, for each $i \in |\mathbb{J}|$ take $F(X_i) \xrightarrow{F(m_i)} F(\sigma_i)$ to be a pointwise subset of $F(\sigma_i)$, and take limits in $\mathsf{Set}^{\mathbb{C}^o}$ to be given pointwise. Then the functor $\bar{F}$ of Theorem 32 is unique. In this case we call the functor a σ-interpretation by analogy with definition 12, and write it $[\![\,]\!]$.*

The category $\mathbb{C}_{\mathsf{P}}[\sigma]$: We have given two constructions, one yielding a category $\mathbb{C}_{\mathsf{P}}$ of substitutions corresponding to proofs with respect to program P, and one yielding arrows that serve as proof steps for data information encapsulated in the index category $\mathbb{J}$ and the functor σ. Finally we define a category of σ-proofs $\mathbb{C}_{\mathsf{P}}[\sigma]$ by merging the two constructions. The objects are pairs (A, S), A in $|\mathbb{C}|$ and S a subfunctor of $\mathbb{C}(A, \sigma(-))$ closed under *both* the conditions 5 of definition 17 (clausal closure) and that of definition 27 ($\mathcal{D}$-closure). Arrows are induced by labels from $\mathbb{C}$ as before.

The category $\mathbb{C}_P^{SLD}[\sigma]$: Given the data $\mathbb{C}$, $\mathbb{J}$ and σ, we define an (unordered) σ-goal to be an object of $\mathbb{C}[\sigma]$, a σ-clause to be a triple $\langle A, G, X(t)\rangle$, where $X(t)$ is an atomic σ-goal (a pullback of a generic σ-predicate) and G a σ-goal, both of sort $A \in |\mathbb{C}|$, and a σ-program P a set of σ-clauses. We can construct a category of generalized resolution proofs directly from $\mathbb{C}_{\mathsf{P}}[\sigma]$ and P by taking, as objects, the objects of $\mathbb{C}[\sigma]$, and as arrows, triples

$$\langle A, S\rangle \xrightarrow{h} \langle B, T\rangle$$

where $\langle A, \langle S\rangle_{\mathcal{D}}\rangle \xrightarrow{h} \langle B, \langle T\rangle_{\mathcal{D}}\rangle$ is an arrow in $\mathbb{C}_{\mathsf{P}}[\sigma]$, ($\langle S\rangle_{\mathcal{D}}$ being the $\mathcal{D}$-closure of S), and composition is given by composition of labels from $\mathbb{C}$. In other words, $\mathbb{C}_P^{SLD}[\sigma]$ is the object part of the comma category (κ, κ) induced by the functor $\kappa : \mathbb{C}[\sigma] \longrightarrow \mathbb{C}_{\mathsf{P}}[\sigma]$ which maps objects $\langle A, S\rangle$ to $\langle A, \langle S\rangle_{\mathcal{D}}\rangle$ and labels to themselves. In this category, arrows correspond to SLD-resolutions induced by program P, weakening, and proof steps induced by the datatype information encoded in σ. We then have soundness and completeness in the following sense

Theorem 34 *Let P be a σ-program and G be a σ-goal. $[\![G]\!]$ is true for every σ-interpretation $[\![\,]\!]$ that is a model of P if and only if there is an arrow $\Box_A \xrightarrow{id_A} G$ in $\mathbb{C}_P^{SLD}[\sigma]$.*

Proof[. sketch We sketch the completeness half of the theorem. Soundness is a straightforward but lengthy induction on the stages of the closure conditions defining objects in $\mathbb{C}_P^{SLD}[\sigma]$.

Let $[\![\,]\!]^s\mathbb{C}[\sigma] \longrightarrow \mathsf{Set}^{\mathbb{C}^o}$ be defined by

$$[\![(A,S)]\!]^s(B) = \{B \xrightarrow{\theta} A \in \mathbb{C} \,|\, \Box_B \xrightarrow{\theta} (A,S) \in \mathbb{C}_P^{SLD}[\sigma]\}$$

where $\Box_B$ is the object $\langle B, \ll \emptyset \gg\rangle$, and on arrows $[\![(A,S)]\!]^s(B \xrightarrow{f} B') = \mathbb{C}(\_, f)$. Then $[\![\,]\!]^s \circ \iota$ is the Yoneda embedding. $[\![\,]\!]^s$ is also a model of the program P. If $G \Rightarrow X(t)$ is a clause of sort A then $G \xrightarrow{id_A} X(t)$ is an arrow in $\mathbb{C}_P^{SLD}[\sigma]$, hence if $\Box_B \xrightarrow{\theta} G$ is in $\mathbb{C}_{\mathsf{P}}[\sigma]$, θ is a label in $[\![X(t)]\!]^s(B)$, so $[\![G]\!]^s(B) \subseteq [\![X(t)]\!]^s(B)$. Let G be a goal of sort A, i. e. an object (A,S) of $\mathbb{C}[\sigma]$. If $[\![G]\!]$ is true for every σ-interpretation $[\![\,]\!]$ that is a model of P, then $[\![G]\!]^s = \mathbb{C}(\_, A)$, s, in particular $A = A \in [\![G]\!]^s(A)$ and $\Box_A \xrightarrow{id_A}$ is in $\mathbb{C}_P^{SLD}[\sigma]$.

□

4 Lifting Data Types to Predicates and Proofs

In this section we give several examples of the use of the machinery developed in preceding sections to incorprate datatype definitions into logic programs.

We take the following syntax for a datatype declaration.

```
datatype 'a foo = c0|c1 ... |k1 of E_1('a)| ... |kn of E_n('a)
```

with `'a` possibly a sequence of variables, and where each E_i(`'a`) is a type term, e.g. `'a * 'a list`, and `ki` the appropriate constructor (e.g. `cons`).

The category $\mathbb{C}$: We assume a finite product category $\mathbb{C}$ of data has been specified with objects corresponding to all required sorts together with an endofunctor $\mathsf{foo}\colon \mathbb{C} \longrightarrow \mathbb{C}$ and endofunctors $E_1, \ldots, E_n$ induced by the type terms E_i in the datatype declaration (e.g. $\lambda X.X \times \mathsf{list}\ (X)$) such that:

1. For each constant $\mathtt{c}_i$ in the datatype declaration there is a natural transformation $\lambda X.\mathbf{1} \xrightarrow{(c_i)} id$.
2. For each constructor $\mathtt{k}_i$ there is a natural transformation k_i from E_i to foo
3. The object $D(\alpha) = \mathbf{1} + \ldots + \mathbf{1} + E_1(\alpha) + \ldots E_n(\alpha)$ exists (as a coproduct) in $\mathbb{C}$, for every object α of $\mathbb{C}$.
4. The arrow
$$D(\mathsf{foo}\ \alpha) \xrightarrow{[c_0 \cdots c_m\_k_1 \cdots k_n]\mathsf{foo}\ \alpha} \mathsf{foo}\ \alpha$$
is an initial algebra in $\mathbb{C}$ for every object α.

The techniques for building the appropriate category $\mathbb{C}$ are well known, and discussed in the literature, so we will not dwell upon this question here. The reader should consult e.g. [20, 13, 14, 23, 3].

Definition 35 (The Index category $\mathbb{J}$) *Let $\mathcal{S} = \{\gamma_1, \ldots, \gamma_n\}$ be the* multi-*set of sorts of predicates in the program (some sorts may be repeated) and let $\mathbb{N}$ be the preorder on the set $\{1, \ldots, n\}$ such that there is precisely one arrow of each type, i.e. all objects of $\mathbb{N}$ are isomorphic. $\mathbb{J}$ will be the smallest subcategory of $\mathbb{C} \times \mathbb{N}$ subject to the following conditions:*

1. *$(\mathbf{1}_{\mathbb{C}}, l), (\gamma_l, l) \in |\mathbb{J}|$ for all $l = 1, \ldots, n$;*
2. *for each $(\alpha, l) \in |\mathbb{J}|$ and $l' = 1, \ldots, n$, we have*
$$!_{(\alpha,l)} \in \mathbb{J}((\alpha, l), (\mathbf{1}_{\mathbb{C}}, l'));$$
3. *for each $(h, f) \in \mathbb{J}((\alpha, l), (\beta, l'))$, the following diagrams are included in $\mathbb{J}$;*
$$(\mathsf{foo}(\alpha), l) \xrightarrow{(\mathsf{foo}(h), f)} (\mathsf{foo}(\beta), l') \qquad (7)$$
4. *for each constant ci in the declaration, $(\alpha, l) \in |\mathbb{J}|$, and $l' = 1, \ldots, n_j$,*
$$(ci, l' \Rightarrow l) \in \mathbb{J}((\mathbf{1}_{\mathbb{C}}, l'), (\mathsf{foo}(\alpha), l)),$$
where $l' \Rightarrow l$ is the unique arrow in $\mathbb{N}(l', l)$;
5. *for each constructor kj in the datatype declaration and object $(\alpha, l) \in |\mathbb{J}|$, all product diagrams*
$$(Ej(\alpha), l) \xrightarrow{\pi_k} (Ej^{(k)}(\alpha), l) \qquad (1 \le k \le n_j) \qquad (8)$$
are included in $\mathbb{J}$.

Given a $\mathbb{J}$-span

$$\begin{tikzcd} & (\beta, l') \arrow[dl, "f_1"'] \arrow[dr, "f_{n_j}"] & \\ (Ej^{(1)}(\alpha), l) & \dots & (Ej^{(n_j)}(\alpha), l) \end{tikzcd} \tag{9}$$

it must be the case that

$$\langle f_1, \dots, f_{n_J} \rangle \in \mathbb{J}((\beta, l'), (Ej(\alpha), l)).$$

The special products in $\mathcal{D}$ are precisely:

1. *the terminators $(\mathbf{1}_{\mathbb{C}}, l)$; and*
2. *the products of Diagram 8.*

Let $\sigma : \mathbb{J} \longrightarrow \mathbb{C}$ be the projection functor onto the first coordinate.

The construction implements the appropriate extension of SLD resolution in the following sense.

Theorem 36 *Given the data $\mathbb{J} \xrightarrow{\sigma} \mathbb{C}$ described above, the category $\mathbb{C}_{\mathsf{P}}[\sigma]$ contains all instances of the following arrows, for $\gamma \in \mathcal{S}$ and each constructor k_i:*

- $X_{E_i(\gamma)} \xrightarrow{(k_i)_\gamma} \mathsf{foo}(X_\gamma)$ *(where k_i is a label from $\mathbb{C}$)*
- $X_{E_i(\gamma)}(\langle t_1, \dots, t_n \rangle) \xrightarrow{id_\gamma} \mathsf{foo}\ \mathsf{X}_\gamma(k_i(t_1, \dots, t_n))$
- *All SLD proofs over the Horn Clauses in P.*

Observe that the predicate $\mathsf{foo}(X_\gamma)$ means $X_{\mathsf{foo}\gamma}$, which in the category $\mathbb{C}_{\mathsf{P}}[\sigma]$ is $(\mathsf{foo}\gamma, \ll id_{\mathsf{foo}\gamma} \gg)$. It is easily shown foo extends to an endofunctor on $\mathbb{C}_{\mathsf{P}}[\sigma]$ (The case where foo is a monad is studied in [17]).

In the case of e.g. the list datatype, the arrow $X_{E_1(\gamma)} \xrightarrow{k_1} \mathsf{foo}\ \mathsf{X}_\gamma$ becomes $X_{\gamma \times \mathsf{list}(\gamma)} \xrightarrow{cons} \mathsf{list}(X_\gamma)$ and $X_{E_i(\gamma)}(\langle t_1, \dots, t_n \rangle) \xrightarrow{id_\gamma} \mathsf{foo}(X_\gamma)(k_i(t_1, \dots, t_n))$ becomes $X_{\gamma \times \mathsf{list}(\gamma)}(\langle t_1, t_2 \rangle) \xrightarrow{cons} \mathsf{list}(X_\gamma)(cons(t1, t2))$ which corresponds to the the proof rule

$$\mathsf{list}(X_\gamma)(cons(t1, t2)) \overset{id}{\leadsto} X_\gamma(t1), \mathsf{list}(X_\gamma)(t_2)$$

4.1 Encapsulation

Finally we show how to lift a datatype definition with member functions

```
datatype 'a foo = c0|c1 ... |k1 of E_1('a)| ... |kn of
  E_n('a)
begin
  fun f1 ...
  ...
end
```

where the functions between the `begin...end` are arrows $\alpha \xrightarrow{f} \rho$ in the original term category $\mathbb{C}$.

All we need to change is the definition of $\mathbb{J}$, which now must be a disjoint union of the $\mathbb{J}_\gamma$ and the diagrams $\alpha \times \alpha \xrightarrow{id_\alpha \times f} \alpha \times \rho$. The generic predicate $m_j \equiv X_j \xrightarrow[Id_{\sigma_j}]{} (\alpha \times \rho \ll id_{\alpha \times \rho} \gg)$ resulting from this extra component of $\mathbb{J}$ is called `pred[f]` in the program syntax. It will automatically satisfy the condition that `pred[f](X,f(X))` is an isomorphism in the categories $\mathbb{C}_{\mathsf{P}}[\sigma]$ and $\mathbb{C}_P^{SLD}[\sigma]$. That is to say, the construction "hard-wires" the graph of the function f into the binary relation `pred[f]`. This makes it possible for an implementation to supply the predicate without supplying the function, which would remain hidden. When interpreted in any model `pred[f](X,f(X))` will be true, and in any resolution proof in $\mathbb{C}_P^{SLD}[\sigma]$ this goal is isomorphic to the empty goal of its sort. We illustrate with an example based on the code in the introduction. We consider the `length` function given in the fragment

```
begin module "list"
datatype 'a list =  nil | cons of 'a * 'a list
  begin
    fun length nil = (0:int)
     |  length (cons (a,t)) = 1 + length t
  end;
```

which will give rise to an arrow $\mathsf{list}(\gamma) \xrightarrow{length} int$ in $\mathbb{J}$ (and in $\mathbb{C}$), and to the arrow

$$\mathsf{list}(\gamma) \xrightarrow{\langle id, length\rangle} \mathsf{list}(\gamma) \times int$$

in $\mathbb{J}$, mapped by σ to itself in $\mathbb{C}_{\mathsf{P}}[\sigma]$. The objects in category $\mathbb{C}_{\mathsf{P}}[\sigma]$ are pairs (A, S) where S must satisfy the closure requirement:

$$\textit{for every arrow} A \xrightarrow{\varphi} \mathsf{list}(\gamma), \varphi \circ \langle id, length\rangle \textit{ is in } S(\mathsf{list}(\gamma) \times int) \tag{10}$$

since S is a subfunctor of $\mathbb{C}(A, \sigma(\_))$. The predicate `pred[length]` is represented in $\mathbb{C}_{\mathsf{P}}[\sigma]$ and in $\mathbb{C}_P^{SLD}[\sigma]$ as the generic

$$(\mathsf{list}(\gamma) \times int, \ll id \gg) \xrightarrow[id]{} (\mathsf{list}\gamma \times int, \ll \emptyset \gg)$$

Its instantiation (pullback) along $\langle id, length\rangle$ is

$$(\mathsf{list}(\gamma), \ll \langle id, length\rangle \gg) \xrightarrow[id]{} (\mathsf{list}\gamma, \ll \emptyset \gg)$$

which has the inverse $(\mathsf{list}(\gamma), \ll \langle id, length\rangle \gg) \xleftarrow[id]{} (\mathsf{list}\gamma, \ll \emptyset \gg)$ since, perhaps suprisingly at first, $\ll \langle id, length\rangle \gg \subseteq \ll \emptyset \gg$ by condition (10) with $A = \mathsf{list}(\gamma)$ and φ the identity. Therefore the (reverse SLD step corresponding to the) arrow $\square_\sigma \xrightarrow{id}$ `pred[length]`$(\langle id, length\rangle)$ is in $\mathbb{C}_P^{SLD}[\sigma]$.

5 Conclusion and future work

We have described a construction of a category of predicates over a base category $\mathbb{C}$, generic up to satisfaction of program clauses and datatype definitions, which gives a category of non-deterministic resolution proofs that implement extended resolution steps capturing program, datatype and base-category information. The framework is proposed here as a blueprint for incorporating certain types of extensions into constraint logic programming. It has been used elsewhere to extend Hereditarily Harrop logic programming with constraints on terms [10, 7] and with monads [17]. We hope it will prove a natural vehicle for declarative approach to the *limited* amounts of control and state that are needed in logic programming as well.

Conspicuously absent in this study is a systematic approach to a deterministic implementation of an abstract machine based on this framework, a set of categorical narrowing rules for execution of the code, and reduction rules for computation of pullbacks (or a canonical choice of unifiers) in the base category $\mathbb{C}$, when $\mathbb{C}$ is suitably chosen (for example with recursively enumerable pullbacks).

References

[1] A. Asperti and S. Martini. Projections instead of variables, a category theoretic interpretation of logic programs. In *Proc. 6^{th} ICLP*, pages 337–352. MIT Press, 1989.

[2] R. Barbuti, M Bellia, and G. Levi. *Leaf: a Language which Integrates Logic Equations and Functions*. Prentice-Hall, 1984.

[3] Michael Barr and Charles F. Wells. *Category Theory for Computing Science*. Prentice-Hall International, Englewood Cliffs, NJ, USA, 1990.

[4] A. Corradini and A. Asperti. A categorical model for logic programs: Indexed monoidal categories. In *Proceedings REX Workshop '92*. Springer Lecture Notes in Computer Science, 1992.

[5] Andrea Corradini and Ugo Montanari. An algebraic semantics for structured transition systems and its application to logic programs. *Theoretical Computer Science*, 103:51–106, 1992.

[6] Roy Crole. *Categories for Types*. Cambridge University Press, 1993.

[7] Mary de Marco. *Dynamic Constraints and Hereditarily Harrop Programming*. PhD thesis, Wesleyan University, 1999.

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

[9] Stacy E. Finkelstein, Peter Freyd, and James Lipton. Logic programming in tau categories. In *Computer Science Logic '94, LNCS 933*. Springer, 1995.

[10] Stacy E. Finkelstein, Peter Freyd, and James Lipton. A new framework for declarative programming. *Theoretical Computer Science*, To appear. Technical report available as `www.cs.wesleyan.edu/~lipton/jftp/dist96-nfdp.ps`.

[11] Peter Freyd and Andre Scedrov. *Categories, Allegories*. North-Holland, 1990.

[12] Joseph Goguen and José Meseguer. *Equality Types and Generic Modules for Logic Programming*. Prentice-Hall, 1984.

[13] Tatsuya Hagino. *A Category Theoretic Approach to Data Types*. PhD thesis, University of Edinburgh, Department of Computer Science, 1987. CST-47-87 (also published as ECS-LFCS-87-38).

[14] Bart Jacobs and Jacob Rutten. *A Tutorial on (Co)algebras and (Co)induction*, volume 62, pages 222–259. 1997.

[15] M. Martelli M. Falaschi, G. Levi and C. Palamidessi. Declarative modeling of the operational behavior of logic languages. *TCS*, 69(3), 1989.

[16] Michael Makkai and Gonzalo Reyes. *First Order Categorical Logic*, volume 611 of *Lecture Notes in Mathematics*. Springer-Verlag, 1977.

[17] R. McGrail. *Modules,Monads and Control in Logic Programming*. PhD thesis, Wesleyan University, 1998. To appear.

[18] F. Orejas, E. Pino, and H. Ehrig. Algebraic methods in the compositional analysis of logic programs. In P. Ruzicka I. Privara, B. Rovan, editor, *Proc. MFCS 94*, volume 841 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[19] P. Panangaden, V. Saraswat, P.J. Scott, and R.A.G. Seely. A hyperdoctrinal view of constraint systems. In *Lecture Notes in Computer Science 666*. Springer Verlag, 1993.

[20] A. Poigné. Algebra categorically. In *Category Theory and Computer Programming*. Springer, 1986.

[21] J. Power and Y. Kinoshita. A new foundation for logic programming. In *Extensions of Logic Programming '96*. Springer Verlag, 1996.

[22] D. Pym. Functorial kripke models of the $\lambda\pi$-calculus. Lecture at Newton Institute Semantics Programme, Workshop on Category Theory and Logic Programming, Cambridge, September 1995, 1995.

[23] H. Reichel. *Initial Computability, Algebraic Specifications, and Partial Algebras*. Oxford University Press, 1987.

[24] D.E. Rydeheard and R.M. Burstall. A categorical unification algorithm. In *Category Theory and Computer Programming*, 1985.