Encapsulating data in Logic Programming via Categorical Constraints

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

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 the base syntactic category in which arrows correspond to resolution proofs subject to the specified data constraints.

1 Introduction

Declarative programming languages have proliferated in the quest for languages with greater expressive power and efficiency which maintain at least a semblance of the original declarative commitment: programs are executable specifications. Some logic programming constructs compromise this executable specification property, often called “declarity”, for the sake of efficiency, but much research is aimed at placing more of the control components into the logic itself, possibly by expanding the scope and definition 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 permits semantical treatment in essentially the same way as 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.

We propose expanding the specification formalism in a way that admits direct definitions of data and constraint information that are already mathematically clear, or verified elsewhere using appropriate tools.
We consider a datatype definition 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 [8], and similar in spirit to, e.g., [1-3,4,27]. Although the framework admits extensions to Hereditarily Harrop or Higher-order logic, we will only consider Horn logic here.

### 1.1 An Example

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

```haskell
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;

?- list even (cons(2,(cons(4,(cons(6,nil))))))).
- yes

?- length_of((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 +,*,/-:int -> int, constants 0,1,2,...:int and equational constraints are defined elsewhere.

The predicate `length_of` 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. The query

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

will yield the solution 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 generalized unification with constraints:

```
length_of((cons (2,(cons (4,(cons (5,nil)))))),A+2) \Rightarrow \Box.
```

and with new proof steps that incorporate data type information.
list X (cons(a,t)) \mapsto X(a), list X (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.

1.2 Related Work

Module proposals, and data type definitions for logic programming have ap-
peared since the early 1980's, notably [11, 2]. There have been scores of at-
tempts to incorporate functions and types into logic programming since then,
using narrowing (see e.g. [16, 13, 22]) and via extensions to the logic [20, 19, 23]).
Our contribution here is to introduce a new technique for lifting certain con-
straints on data to predicates on that data and proofs between them, in the
general framework of a categorically defined constraint signature.

Categorical approaches to logic programming features appeared in the mid
1980's in Rydeheard and Burstall's categorical treatment of unification [20] 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 Mont-
tanari [1, 4, 3], Panangaden, Scott, Seely, Saraswat, [25], Power and Kinoshiba
[27] Diaconescu [7] Pym [28], Orejas, Ehrig and Pino [24], and Finkelstein, Freyd
and Lipton [8, 9].

2 Logic Programming Categories

In the remainder of this paper, we make use of the basic categorical framework
defined in [8]. The framework is similar to that introduced by other authors,
e.g. [1], to formalize logic programming categorically, and based on standard
approaches to categorical logic, e.g. [17]. It is briefly sketched here.

2.1 Categorical Logic

In a nutshell, the categorical representation of the syntax of logic programs
is as follows: an FP category (category with finite products) \( \mathcal{C} \) 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
\( (\sigma, \rho) \) (arrows \( \sigma \to \rho \)), and constants of sort \( \rho \) (arrows of the form \( 1 \to \rho \),
where \( 1 \) 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.

A certain category represents the conventional Herbrand Universe over a one-sorted signature. In the so-called the Lawvere Algebraic Theory for a one-sorted signature with at least one constant (considered in e.g. [8]) and with no equations, 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 allows us to build-in constraint information and data types into the syntax. 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) is generalized to finding a pair of substitutions (arrows) with a common source \( \theta \), \( \psi \) 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\(^1\). In extensions of Horn logic, such as \( \lambda \)-prolog, queries and programs may share variables, which requires explicit sharing of domains. Unification then reduces to equalizing the arrows (see \([9]\))

This idea of unification over a category, which opens up the door to a sweeping categorical generalization of the syntax and proof theory of logic programming, was suggested (in a more or less equivalent form) in the mid-1980’s by several authors, e.g. Goguen, Burstall and Rydeheard, and is the basis of the treatment in \([8,9,1,27]\) among others.

For basic concepts of category theory, we refer the reader to \([10]\), and for the elements of indexed category theory, see \([5]\). We note that compositions of arrows are written in diagrammatic order: \( A \xrightarrow{f} B \) composed with \( B \xrightarrow{g} C \) is \( A \xrightarrow{fg} C \).

### 2.2 Generic Predicates and Unordered Goals

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

\[
\begin{array}{ccc}
R(t) & \to & R \\
\downarrow & & \downarrow \\
\alpha & \to & \sigma \\
\end{array}
\]

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\(^2\) and they should not be chosen to conflict

---

\(^1\) If the ambient category is the Herbrand Category, the occurs-check will be automatically enforced.

\(^2\) 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.
with the meaning of the logic program in which they will be used. Unless 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.
We will now make this idea precise below and show how to construct them.

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

- For every arrow $t$ in $D$ the pullback $t^!(X)$ exists.
- No such pullback is an isomorphism.

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

- **objects**: pairs $\langle A, S \rangle$ where $A \in \mathcal{C}$ and $S$ is a sequence $S_1, \ldots, S_n$ of finite sets $S_i \subseteq \text{Hom}_\mathcal{C}(A, b_i)$.
- **arrows**:  are triples $\langle A, S \rangle \xrightarrow{f} \langle B, T \rangle$ where $A \xrightarrow{f} B$ is an arrow in $\mathcal{C}$ and $fT \subseteq S$, that is to say, for every $i$, $(1 \leq i \leq n)$ and every $t \in T_i$, $f t_i \in S_i$.

  The arrow $f$ in $\mathcal{C}$ is called the **label** of $\langle A, S \rangle \xrightarrow{f} \langle B, T \rangle$. Composition of arrows is inherited from $\mathcal{C}$. 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 $\mathcal{C}$.

Given an object $\langle A, S \rangle$ we will use the notation $tS$, where $t$ is an arrow in $\mathcal{C}$ 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 $\mathcal{C}[X_1, \ldots, X_n]$, may have an identity arrow in $\mathcal{C}$ as a label, and not even be an isomorphism in $\mathcal{C}[X_1, \ldots, X_n]$. We will be paying special attention to a certain class of such arrows.

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

- a terminal object $\langle 1, \emptyset \rangle$, where $\emptyset$ 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 $A \xrightarrow{\pi_1} A \times B \xrightarrow{\pi_2} B$ is a product in $\mathcal{C}$.

Furthermore, the functor $\mathcal{C} \xrightarrow{t} \mathcal{C}[X_1, \ldots, X_n]$ given by mapping objects $A$ to $\langle A, \emptyset \rangle$ and arrows $A \xrightarrow{f} B$ to $\langle A, \emptyset \rangle \xrightarrow{f} \langle B, \emptyset \rangle$, is a limit-preserving, full embedding. Limit preservation follows from the fact that $t$ 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. $\mathbb{C}[X_1, \ldots, X_n] \cong \mathbb{C}[X_1 \cdots X_n]$.

2. Let $\sigma$ be a permutation of the first $n$ positive integers. Then $\mathbb{C}[X_1, \ldots, X_n] \cong \mathbb{C}[X_{\sigma(1)}, \ldots, X_{\sigma(n)}]$.

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 $\langle b_i, J^i \rangle \xrightarrow{id_{b_i}} \langle b_i, \emptyset \rangle$, where the $J^i$ are the "basis vectors"

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

Theorem 6 The indeterminate subobjects $X_i$ of $b_i$ are generic with respect to the maps in the image of $\text{Hom}_{\mathbb{C}}(\_ b_i)$ under $\mathbb{C} \xrightarrow{t} \mathbb{C}[X_1, \ldots, X_n]$.

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

$$
\begin{array}{ccc}
\langle A, tJ^i \rangle & \xrightarrow{id_A} & \langle b_i, \{id\} \rangle \\
\downarrow id_{b_i} & & \downarrow id_{b_i} \\
\langle A, \emptyset \rangle & \xrightarrow{t} & \langle b_i, \emptyset \rangle
\end{array}
$$

so $X(t) = \langle A, tJ^i \rangle \xrightarrow{id_A} \langle A, \emptyset \rangle$ 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 $id_A t \in \emptyset$. □

Definition 7 An object $\langle A, H \rangle$ is atomic if $H$ is of the form $tJ^i$ for a basis vector $J^i$ and some arrow $A \xrightarrow{t} \sigma_i$. In conventional notation $\langle A, H \rangle$ is the formula $X_i(t)$.

If $A$ is an object of $\mathbb{C}$, we say that the monic $\langle B, S \rangle \xrightarrow{f} \langle A, \emptyset \rangle$ is a canonical (representative of a) subobject of $\langle A, \emptyset \rangle$ 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" $\langle A, \emptyset \rangle$. This allows us to define a natural indexed structure [5] for $\mathbb{C}[X_1, \ldots, X_n]$ over $\mathbb{C}$.

Definition 8 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 $\langle A, S \rangle \xrightarrow{id_A} \langle A, \emptyset \rangle$, and with morphisms given by arrows between their sources labelled by the identity on $A$ in $\mathbb{C}$. Then let $p : \mathbb{C} \longrightarrow \text{CAT}$ be the strict indexed category given by
\[ p(A) = \mathbb{C}_A[X_1, \ldots, X_n] \]

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

Notice that the pullback operation referred to by \( f^\# \) maps \( \langle B, S \rangle \xrightarrow{\text{id}_B} \langle B, \emptyset \rangle \) to \( \langle A, fS \rangle \xrightarrow{\text{id}_A} \langle A, \emptyset \rangle \). Thus \((fg)^\#\) is precisely \( g^\# f^\#\) on the nose. Now we define a “canonical intersection” functor for the indexed category \( p \).

**Definition 9** 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 pullbacks, 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 10** Every object \( \langle A, S \rangle \) is representable as (i.e., equal on the nose to) the canonical intersection

\[ \bigcap \{t^\#(X_i) : t \in S_i, 1 \leq i \leq n\} \]

where the pullbacks are canonical: \( t^\#(X_i) = \langle A, t_i^0 \rangle = \langle A, \emptyset \cdots \emptyset \{t\} \emptyset \cdots \emptyset \rangle \).

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

**Theorem 11 (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 \( 1 \leq i \leq n \). 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 \( F_B : \mathbb{C}[X_1, \ldots, X_n] \longrightarrow \mathbb{D} \), unique up to isomorphism, such that the following diagram commutes.

\[ \begin{array}{ccc}
\mathbb{C}[X_1, \ldots, X_n] & \xrightarrow{F_B} & \mathbb{D} \\
\downarrow & & \uparrow \\
\mathbb{C} & \xrightarrow{F} & \mathbb{D}
\end{array} \]

\( F_B \) is called the evaluation functor induced by the \( B_i \).

**Proof.** Define \( F_B \) on objects by \( F_B(\langle A, S \rangle) = \lim_i \{F(t_i^\#(B_i)) : t \in S_i, 1 \leq i \leq n\} \). 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 \( F_B(\langle A, S \rangle) \), the limit of the family of monics \( \{F(t_i^\#(B_i)) : t \in S_i, 1 \leq i \leq n\} \) targeted at \( F_A \), 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 $\{F(t)(B_i) : t \in S_i, 1 \leq i \leq n\}$. There is therefore a unique induced arrow $F(A, S) \rightarrow F(A', S')$ which is the value of $F(A, S) \rightarrow (A', S')$. The details, and those of the proof of limit preservation, are left to the perseverant reader.

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

**Corollary 12** Assume the category $\mathcal{D}$ in the preceding theorem is $\text{Set}^{co}$ and that $F$ is the Yoneda embedding. Choose the sequence of subobjects $B_i$ of $Fb_i = \text{Hom}_{\mathbb{C}}(\_b_i)$ to be canonical, that is to say, pointwise subsets of $Fb_i$, and take limits in $\text{Set}^{co}$ to be given pointwise (not just up to isomorphism, but on the nose). Then the evaluation functor $F_B$ of the preceding theorem is unique.

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

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

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 (10) above, every object $\langle A, S \rangle$ in $\mathbb{C}[X_1, \ldots, X_n]$ is an intersection

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

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 11. Since the $S_i$ are sets, they cannot capture order or repetitions of atoms within goals. An ordered counterpart will be defined below.

**Definition 13** 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 $\text{Hom}_{\mathbb{C}}(\_b_i)$ as in corollary 12.

In other words, an interpretation is a functor

$$[] : \mathbb{C}[X_1, \ldots, X_n] \rightarrow \text{Set}^{co}$$

- 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.
2.3 Proof 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 $\sigma_i$, as in the construction of the categories in the preceding section. We will refer to objects $X_i(t_i) = t_i\sigma_i$ of sort $A_i$, where $A \xrightarrow{t} \sigma_i$ is an arrow in $\mathbb{C}$, as atomic goals or predicates (of sort $A_i$) objects $\langle A, S \rangle$ of $\mathbb{C}[X_1, \ldots, X_n]$ as unordered (or non-deterministic) goals (of sort $A$), and sequences $X_i(t_1), \ldots, X_m(t_m)$, where each atomic predicate $X_i(t_j)$ is of sort $A_i$, as ordered goals of sort $A$. Note that there is a “forgetful” function $\beta$ which takes ordered goals to the underlying unordered ones:

$$\beta(X_i(t_1), \ldots, X_m(t_m)) = \langle A, S \rangle,$$

(3)

where $S_k = \{t_j : i_j = k\}$ that is to say, the set of all terms occurring as arguments to the $k$th generic predicate, wherever it occurs (0 or more times) in $X_i(t_1), \ldots, X_m(t_m)$. 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 $\text{def}^\#_i$, which removes one occurrence (the first one, in the ordered case) of the atomic goal $X_i(t)$ if it exists, and returns the original goal unchanged, otherwise. More formally, in the unordered case: $\text{def}^\#(A, S) = \langle A, T \rangle$ where $T_i = S_i$ if $i \neq k$ and $T_k = S_k \setminus \{t\}$.

**Definition 14** 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 $\beta$ in (3) extends naturally to a map from ordered clauses to unordered ones.

**Definition 15** 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 $\theta$, and program clause $c = \langle A, H, X_k(u) \rangle$, denoted

$$G_1 \xrightarrow{\theta \cdot c} G_2$$

is a 4-ary relation (relating $G_1, \theta, c, G_2$) 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, $G_1 = G, X_k(t), \sigma'$,
- there is an arrow $A_2 \xrightarrow{\psi} \sigma'$ 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 \( \text{SLD-step} \) \( \langle A_1, S \rangle \xrightarrow{\theta} \langle A_2, T \rangle \) between them exists when a clause \( c \) and unifying arrows \( \theta \) and \( \psi \) exist, as above, and \( T = \theta(\text{def} S) \cap \psi H \). In both the ordered and unordered cases, the arrow \( \theta \) is called the substitution of the SLD step, and \( c \) its clause.

We define an SLD-sequence between goals, \( G \xrightarrow{\theta} \cdots \xrightarrow{\theta} G' \) 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 \( \theta_i \), and \( \theta \) is the composition \( \theta_n \cdots \theta_1 \). The "reflexive case" \( n = 0 \) and \( \theta = \text{id} \) is allowed.

**Definition 16** The category \( \mathbb{C}^{\text{SLD}} \) 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 \( \langle B, T \rangle \xrightarrow{\theta} \cdots \xrightarrow{\theta} \langle A, S \rangle \) is an SLD sequence. We call \( A \xrightarrow{\theta} B \) in \( \mathbb{C} \) the label of this arrow of \( \mathbb{C}^{\text{SLD}} \).

The Category of Proofs \( \mathbb{C}_P \) We now modify the generic predicate construction \( \mathbb{C}[X_1, \ldots, X_n] \) to produce a category of predicates which are generic modulo the clauses in program \( P \). In a sense that will be made precise below, it is the freest category of predicates over \( \mathbb{C} \) satisfying the program clauses.

Let \( P \) be a Horn Clause program over \( \mathbb{C}[X_1, \ldots, X_n] \) of sort \( \alpha \), that is to say a set of clauses \( \{c_1, \ldots, c_m\} \) each of the form

\[
X_{i_1}(t_1), \ldots, X_{i_n}(t_n) \Rightarrow X(t)
\]

(4)

where \( X \) is a generic predicate of sort \( \sigma \), \( \alpha \xrightarrow{\sigma} \alpha \) an arrow in \( \mathbb{C} \), each \( X_{i_k} \) a predicate of sort \( \sigma_{i_k} \), and each \( \alpha \xrightarrow{\sigma_{i_k}} \alpha \) an arrow in \( \mathbb{C} \).

**Definition 17** Let \( \alpha \) 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 \( \alpha \) is said to be closed under \( T \) if \( T(S) = S \). \( S \) is generated with respect to \( T \) if \( S = T(U) \) for a family \( U \). We will write \( \langle U \rangle_T \) for \( T(U) \) the family generated by \( U \), or just \( \langle U \rangle \) when the closure operator is understood.

**Definition 18** Let \( \mathbb{C} \) be a finite-product category, and \( P \) a Horn clause program over \( \mathbb{C}[X_1, \ldots, X_n] \). The category \( \mathbb{C}_P \) 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 finitely generated set of arrows from \( \alpha \) to \( \sigma_i \), with respect to the following closure condition:
  - For each clause (4) of sort \( \alpha \) and each arrow \( A \xrightarrow{\varphi} \alpha \) satisfying
    
    \[
    \varphi t_1 \in S_{i_1}, \ldots, \varphi t_n \in S_{i_n}
    \]
    
    (5)
    
    we must have \( \varphi t \in S_k \) where \( \sigma_k \) is the sort of the head predicate \( X \).

- **arrows** are triples \( (A, S) \xrightarrow{T} (B, T) \) such that \( A \xrightarrow{T} B \) (the label) is an arrow in \( \mathbb{C} \), and \( T \subset S \). Composition is defined by composing labels.
If we let $T_P$ be the operator on sets of arrows with a common source $A$ given by
\[ T_P(S) = \bigcup_{c \in P} \{ \varphi t : A \to \alpha \text{ and } \varphi t_1 \ldots \varphi t_n \in S \} \]
then the requirement on $S$ in the preceding definition is equivalent to being finitely generated with respect to $T_P$.

**Lemma 19** There is a faithful functor $t_P : C[X_1, \ldots, X_n] \to C_P$ given by $(A,U) \mapsto (A,\langle U \rangle)$ on objects, and which preserves labels of arrows.

**Proof.** Since the closure operator $T_P$ is monotone, $t_P$ maps arrows to arrows. Fidelity and functoriality is immediate. \qed

### 2.4 Semantics

Let $P$ be a program, $G^P$ the collection of ordered goals over $P$, and $[ ]$ an interpretation into $\text{Set}^{C^P}$ as in definition (13), and $\beta$ the forgetful map from ordered to unordered goals. Then observe that $[ ]$ is a function $G^P \to \text{Set}^{C^P}$. We sometimes overload the symbol $[ ]$ to denote $\beta[ ]$, since context will always make clear what is meant. We call the composition $t_P : G^P \to \text{C}_P$ the representation of goals in the category $\text{C}_P$.

Note that by corollary (12), the value of $[ ]$ on goals is completely determined by its value on generic predicates.

We now give a quick sketch of the main results on semantics from [8, 9]. We give a categorical analogue of the Kowalski-van Emden bottom-up semantics. Proofs can be found in the cited references, and are, for the most part, omitted.

**Definition 20** An interpretation $[ ]$ is a model of program $P$ if for every clause $cl : \text{hd} \Rightarrow \text{tl}$ we have $[\text{tl}] \subseteq [\text{hd}]$. A goal $G$ of sort $\alpha$ is said to be true in the interpretation if the image $[\beta(G)] \in \text{Set}^{C^P}$ of the monic
\[ \beta(G) \to (\alpha, \emptyset) \]
is an isomorphism.

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

**Lemma 21 (Soundness)** Let $[ ]$ be a model of program $P$. Suppose $G_1$ and $G_2$ are goals, of sorts $\alpha_1, \alpha_2$ respectively, and there is a resolution proof $G_1 \rightsquigarrow \ldots \rightsquigarrow G_2$, where $\alpha_2 \to \alpha_1$ is the computed answer substitution. Then $[G_2] \subseteq [\theta G_1]$. In particular, if $G_2$ is $\Box_{\alpha_2}$ then $\theta G_1$ is true in the model.
We now define a categorical analogue of the $T_P$ operator of Kowalski and Van Emde, an operator $E_P$ on the lattice of interpretations.

**Definition 22** Let $[ ]$ be an interpretation and $X_1, \ldots, X_n$ the sequence of generic predicates in program $P$. Then

$$E_P([ ])(X_i) = \bigcup_{t \in X_i(t) \in P} \text{Im}_{[t]}([d_i])$$

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

In [8,9] 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 23** 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 [8,9]

**Theorem 24 (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 $G \rightsquigarrow \; \cdots \; \rightsquigarrow \Box$. We now establish some properties of the proof-theoretic category $C_P$. Since it is a finite-product category, arrows in $C_P$ are not, strictly speaking, SLD proof steps (in reverse), but a bit more. Arrows in $C_P$ correspond to extended resolution steps that include weakening of goals and instantiation of goals, as well as products of basic SLD steps that can be thought of as parallel resolutions of multiple goals. If we are interested in restricting to pure SLD steps, we can construct the category defined in 16, or construct a monoidal version of $C_P$, taking as objects $(A, S)$ where $S$ is a sequence of arrows instead of a set (or using the free indexed monoidal categories of [4]).

But the present category suffices for our purposes, since the only global proofs $\Box \alpha \rightsquigarrow \beta P(G)$ correspond to real SLD proofs $G \rightsquigarrow \; \cdots \; \rightsquigarrow \Box \alpha$.

**Theorem 25 (Soundness of $\beta \circ \alpha$)** Let $G_1 \rightsquigarrow \; \cdots \; \rightsquigarrow G_2$ be an SLD sequence, and let $(\alpha_1, \langle U_1 \rangle)$ and $(\alpha_2, \langle U_2 \rangle)$ be the images in $C_P$ of $G_1$ and $G_2$ under $\beta \circ \alpha_P$. Then $G_2 \rightsquigarrow \; \alpha \; G_1$ is an arrow in $C_P$.

**Proof.** Observe that the closure condition (5) of definition (18) guarantees that for each clause $t \Rightarrow \text{id}$ of $P$, the arrow $(\alpha, \langle S_t \rangle) \overset{\text{id}}{\longrightarrow} (\alpha, \langle S_{\text{id}} \rangle)$ is in $C_P$, for $\beta_P(hd) = (\alpha, \langle S_{\text{id}} \rangle)$ and $\beta_P(hd) = (\alpha, \langle S_{\text{id}} \rangle)$, the requirement for arrows in $C_P$ being $S_t \subseteq S_{\text{id}}$ 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 $\theta$ targeted at $\alpha (\alpha, \theta(S_t)) \overset{\text{id}}{\longrightarrow} (\alpha, \theta(S_{hd}))$ is also in $C_P$, and that $G_1 \rightsquigarrow \; \cdots \; \rightsquigarrow G_2$ implies $\theta G_1 \rightsquigarrow \; \cdots \; \rightsquigarrow G_2$, it is straightforward to show that any resolution sequence with answer substitution $\theta$ maps to an arrow with label $\theta$ in the opposite direction. \qed
Theorem 26 (Completeness of $\beta \circ \rho$) If $\square_\alpha \overset{\theta}{\longrightarrow} \beta \rho G$ is an arrow in $C_P$, then there is an SLD proof $G \overset{\theta}{\longrightarrow} \cdots \overset{\theta}{\longrightarrow} \square_\alpha$.

Proof. The empty goal of sort $\alpha$ is represented by the object $(\alpha, \emptyset)$ in $C_P$. Suppose $\beta \rho G = (\gamma, \langle U \rangle)$. That is to say $G$ is of the form $X_{\beta_1}(t_1), \ldots, X_{\beta_n}(t_n)$ and $U$ is

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

Then if $\square_\alpha \overset{\theta}{\longrightarrow} \beta \rho G$ is an arrow in $C_P$, $\theta(U) \subseteq \langle \emptyset \rangle$, that is to say, by the remarks about $T_P$ following definition (18), every $X_{\beta_i}(t_j)$ is true in the Herbrand interpretation $\llbracket \cdot \rrbracket^\ast$ of $P$ for $(1 \leq j \leq n)$. By completeness for $\llbracket \cdot \rrbracket^\ast$ (Theorem 24) there is an SLD proof of $\theta G$ from $P$, hence a proof of $G$ with computed answer substitution $\theta$.

$\square$

3 The Categories $C[\sigma]$ and $C_P[\sigma]$

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.

3.1 $C[\sigma]$

For the duration of this section $C$ is an FP category, $\mathbb{D}$ is a complete category, $J$ is a category, and $D$ is a collection of product diagrams in $J$, which we will call distinguished products, or $D$-products.

Definition 27 ($D$-Product Preservation) We will say that the functor $H : J \longrightarrow \mathbb{D}$ preserves $D$-products if $H$ sends each $D$-product to a product diagram in $\mathbb{D}$.

Also for the rest of this chapter, $\sigma : J \longrightarrow C$ is a functor that is assumed to preserve $D$-products and let $\sigma_1$ denote the $C$-object $\sigma(1)$.

Definition 28 ($D$-Closure) Let $A \in \mathbb{C}$, and $D$ be a set of product diagrams in $C$. A subfunctor $S$ of $C(A, \sigma(-))$ is $D$-closed if it preserves $D$-products. In other words, for every $D$-product diagram for $i_1 \times \cdots \times i_n$, if there are arrows $f_k \in S(i_k)$ for each $k = 1, \ldots, n$, then $\langle f_1, \ldots, f_n \rangle \in S(i_1 \times \cdots \times i_n)$.

In particular, if $1 \in D$, then $\sigma_1 = 1_C$ and $S(1) = \{ !_A \}$.

Definition 29 ($D$-Generating Sets) Let $Y$ be a collection of arrows emanating from $A$, each with target in the range of $\sigma$. $Y$ is said to $D$-generate $S$ if $S$ is the smallest $D$-closed subfunctor of $C(A, \sigma(-))$ containing all the arrows of $Y$. Let $\llbracket Y \rrbracket$ stand for the $D$-closed subfunctor of $C(A, \sigma(-))$ that $Y$ $D$-generates. We call $\llbracket Y \rrbracket$ the $D$-closure of $Y$. 

}
We note that $\ll Y \gg$ may be built up inductively from $Y$ by closing under composition and products.

**Definition 30 ($\mathbb{C}[\sigma]$)** Let $\mathbb{C}[\sigma]$ be defined as follows.

**objects:** pairs $\langle A, S \rangle$ where $A \in \mathbb{C}$ and $S$ is a $D$-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 $hT \subseteq S$, 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 composition 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 $D = \emptyset$ then $\mathbb{C}[\sigma] = \mathbb{C}[X_1, \ldots, X_n]$. Hence Definition 2 is a special case of Definition 30.

**Proposition 31** 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 [18] for details.

**Definition 32** Let $\mathbb{C}(\sigma)$ stand for the full subcategory of $\mathbb{C}[\sigma]$ whose objects are

$$\mathbb{I}(\mathbb{C}) \bigcup \{ \langle \sigma_j \ll \text{Id}_{\sigma_j} \gg \rangle | j \in \mathbb{J} \}.$$

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

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

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

The monic $m_j$ (and by abuse of language, its source $X_j$) will be called the **generic predicate of sort** $\sigma_j$ 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

\[
\begin{array}{ccc}
\langle A, \ll t \gg \rangle & \xrightarrow{t} & X_j \\
\downarrow \text{Id}_A & & \downarrow m_j \\
\langle A, \ll \emptyset \gg \rangle & \xrightarrow{t} & \iota \sigma_j
\end{array}
\]
is a pullback in \( \mathbb{C}[\sigma] \). However, it may be the case that some of these pullbacks are indeed isomorphisms. For example, suppose \( 1 \) is in \( \mathbb{D} \). Then it follows that

\[ X_1 = \sigma \mathbb{1}_1 \text{ so that } X_1 \xrightarrow{m} \mathbb{1}_2 \text{ is iso.} \]

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

**Theorem 33 (Universal Mapping Property II)** Suppose \( \mathbb{D} \) is complete category and \( F : \mathbb{C}(\sigma) \to \mathbb{D} \) is a functor such that

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

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

\[
\begin{array}{ccc}
\mathbb{C}[\sigma] & \xrightarrow{\mathcal{F}} & \mathbb{D} \\
\downarrow & & \downarrow \\
\mathbb{C}(\sigma) & \xrightarrow{F} & \mathbb{D}
\end{array}
\]

commute.

**Proof** (sketch). For \( A \xrightarrow{t} \sigma_j \), let \( \mathcal{F}(t) \) be the pullback of \( X_j \xrightarrow{F(m_j)} F \sigma_j \) along \( F(t) \).

Define \( \mathcal{F}(A, S) \) on objects by \( \mathcal{F}(A, S) = \bigcap \{ \mathcal{F}(t) \mid t \in S(j), j \in [\mathbb{I}] \} \). The right hand side can be shown to be a finite limit, hence it must exist in \( \mathbb{D} \). \( \mathcal{F} \) on arrows is just a consequence of the limit definition of \( \mathcal{F} \) on objects. It follows that \( \mathcal{F} \) has the stated properties. We refer the reader to [18] for the details.

**Corollary 34** Assume the category \( \mathbb{D} \) in Theorem 33 is \( \text{Set}^{\mathbb{C}} \) and that \( F \circ \iota \) is the Yoneda embedding. Also, for each \( i \in [\mathbb{I}] \) take \( F(X_i) \xrightarrow{F(m_i)} F(\sigma_i) \) to be a pointwise subset of \( F(\sigma_i) \), and take limits in \( \text{Set}^{\mathbb{C}} \) to be given pointwise. Then the functor \( \mathcal{F} \) of Theorem 33 is unique.

**The category \( \mathbb{C}_p[\sigma] \)** We have defined two constructions, one yielding a category \( \mathbb{C}_p \) of proofs with respect to program \( P \), and one yielding arrows that serve as proof steps for data information encapsulated in the index category \( \mathbb{I} \) and the functor \( \sigma \), as we shall see in the next section. What remains is to merge the two constructions, which is done by combining the closure conditions on the finitely generated spans \( S \) in the objects \( (A, S) \).

**Definition 35** Let \( \mathbb{C}_p[\sigma] \) be defined as follows.
**objects** are pairs \( \langle A, S \rangle \) where \( A \in [C] \) and \( S \) is a finitely generated family of arrows with common source \( A \) with respect to the following closure conditions:

- \( S \) is a subfunctor of \( C(A, \sigma(\_)) \) such that if
  \[ i_1 \times \ldots \times i_n \xrightarrow{\mu_k} i_k \quad (1 \leq k \leq n) \]
  is a designated product of \( \mathbb{J} \) and for each \( k = 1, \ldots, n \), \( f_k \in S(i_k) \) then
  \[ f_1 \times \ldots \times f_n \in S(i_1 \times \ldots \times i_n). \]

- For each clause
  \[ X_{i_1}(t_1), \ldots, X_{i_n}(t_n) \Rightarrow X(t) \tag{7} \]
  of sort \( \alpha \) in program \( P \), where \( X_{i_n} \) is the generic predicate \( m_{i_n} \) of sort \( \sigma_{i_n} \) of definition (32), and each arrow \( \varphi \alpha \) satisfying
  \[ \varphi t_1 \in S(\sigma_{i_1}), \ldots, \varphi t_n \in S(\sigma_{i_n}) \tag{8} \]
  we must have \( \varphi t \in S_\alpha \) where \( \sigma_\alpha \) is the sort of the head predicate \( X \).

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

### 4 Data Types

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

We assume given a basic category of data \( C_0 \). We take the following syntax for a datatype declaration.

Datatype \( \langle \text{tyvarseq}\rangle \langle \text{ident} \rangle = \langle \text{dataexpn} \rangle \{ 1 \langle \text{dataexpn} \rangle \} \);

where \( \langle \text{tyvarseq} \rangle \) is any sequence of identifiers standing for type variables, and \( \langle \text{dataexpn} \rangle \) is defined by

\[ \langle \text{dataexpn} \rangle ::= \langle \text{ident} \rangle | \langle \text{ident} \rangle \text{ of } \langle \text{tyexpn} \rangle \]

and

\[ \langle \text{tyexpn} \rangle ::= \langle \text{tyconst} \rangle | \langle \text{tyvar} \rangle | \langle \text{tyexpn} \rangle \ast \langle \text{tyexpn} \rangle | \langle \text{tyexpn} \rangle \langle \text{ident} \rangle \]

Some examples:

Datatype 'a list = nil | cons of 'a * 'a list;
Datatype 'a tree = leaf of 'a | node of 'a * 'a tree * 'a tree;

We now discuss how this datatype declaration is formalized in the data \( \mathbb{J} \xrightarrow{\sigma} C \) of \( 3 \). Data encapsulation with member functions will be discussed in the next section.
Formalizing the datatype. We assume given a declaration 
\[ \text{datatype } 'a \text{ foo } = c0 | c1 \ldots | k1 \text{ of } E_1('a) | \ldots | kn \text{ of } E_n('a) \]
where each \( E_i('a) \) is a type term, e.g. 'a * 'a list in the second production for list, and \( ki \) the appropriate constructor (e.g. cons). We also assume an underlying (many-sorted) signature \( \Sigma \) and equational constraints \( E \) specified by the user, and that all type constants in the datatype definition (e.g. string, int, bool) are sorts of \( \Sigma \)

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

1. For each constant \( c_i \) in the datatype declaration there is an arrow \( 1 \xrightarrow{(c_i)} \alpha \) for every object \( \alpha \) in \( \mathbb{C} \) (in fact the family \( \{ (c_i)_{\alpha} \} \) is a natural transformation from the constant functor \( \lambda X.1 \) to \( 1_{\mathbb{C}} \)).
2. For each constructor \( k_i \) there is a natural transformation \( k_i \) from \( E_i \) to \( \text{foo} \).
3. The object \( D(\alpha) = 1 + \ldots + 1 + E_1(\alpha) + \ldots E_n(\alpha) \) exists (as a coproduct) in \( \mathbb{C} \), for every object \( \alpha \) of \( \mathbb{C} \).
4. The arrow 
\[ \frac{D(\text{foo } \alpha)}{\text{foo } \alpha} \]

is an initial algebra in \( \mathbb{C} \) for every object \( \alpha \).

The techniques for building the appropriate algebraic theory \( \mathbb{C} \) are well known, and discussed in the literature, so we will not dwell upon this question here. The reader should consult e.g. [26, 12, 14].

The Irrel category \( \mathbb{J} \):

**Definition 36** Let \( S = \{ \gamma_1, \ldots, \gamma_n \} \) 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. \( (1_c, l), (\gamma_l, l) \in \mathbb{J} \) for all \( l = 1, \ldots, n \);
2. for each \( (\alpha, l) \in \mathbb{J} \) and \( l' = 1, \ldots, n \), we have 
\[ 1_{(\alpha, l)} \in \mathbb{J}((\alpha, l), (1_c, l')) \]
3. for each \( (h, f) \in \mathbb{J}((\alpha, l), (\beta, l')) \), the following diagrams are included in \( \mathbb{J} \):
\[ (\text{foo}(\alpha), l) \xrightarrow{(\text{foo}(h), f)} (\text{foo}(\beta), l') \]

(9)
4. For each constant \( c_i \) in the declaration, \( (\alpha, t) \in \mathbb{I} \), and \( l' = 1, \ldots, n_j \),

\[
(c_i, l' \Rightarrow l) \in \mathbb{J}((1c_i, l'), (\text{foo}(\alpha), l)),
\]

where \( l' \Rightarrow l \) is the unique arrow in \( \mathbb{N}(l', l) \);

5. For each constructor \( k_j \) in the datatype declaration and object \( (\alpha, l) \in \mathbb{I} \), all product diagrams

\[
(E_j^{(\alpha)}(l), l) \xrightarrow{\pi_k} (E_j^{(k)}(\alpha), l) \quad (1 \leq k \leq n_j)
\]

are included in \( \mathbb{J} \).

and given a \( \mathbb{J} \)-span

\[
\begin{array}{ccc}
\langle \beta, l' \rangle & \xrightarrow{f_1} & (E_j^{(\alpha)}(l), l) \\
& & \downarrow f_{n_j} \\
\cdots & & (E_j^{(n_j)}(\alpha), l)
\end{array}
\]

it must be the case that

\[\langle f_1, \ldots, f_{n_j} \rangle \in \mathbb{J}((\beta, l'), (E_j(\alpha), l)).\]

The special products in \( \mathbb{D} \) are precisely:

1. The terminators \( (1c, l) \); and
2. The products of Diagram 10.

Let \( \sigma : \mathbb{J} \longrightarrow \mathbb{C} \) be the functor that forgets the second coordinate.

\( \sigma \) restricted to the full subcategory generated by the objects of the form \( (\alpha, l) \) for a fixed \( l \) is an inclusion into \( \mathbb{C} \). Since there may be more than one copy of a given object \( \gamma \) in the multiset \( S \), \( \sigma \) itself is not necessarily an inclusion.

The cartesian functor \( \sigma \) restricted to any \( \mathbb{J}_\gamma \) is an inclusion into \( \mathbb{C} \). Since there may be more than one copy of a given object \( \gamma \) in the multiset \( S \), \( \sigma \) is not necessarily an inclusion on \( \mathbb{J} \).

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

Theorem 37 Given the index category and functor \( \mathbb{J} \xrightarrow{\sigma} \mathbb{C} \) described above, the category \( \mathbb{C}[\mathbb{P}[\sigma]] \) contains all instances of the following arrows, for \( \gamma \in S \) and each constructor \( k_i \):

- \( X_{E(\gamma)}(k_i) \xrightarrow{(k_i)} \text{foo}(X_{\gamma}) \) (where \( k_i \) is a label from \( \mathbb{C} \))
- \( X_{E(\gamma)}((t_1, \ldots, t_n)) \xrightarrow{\text{id}_{(t_1, \ldots, t_n)}} \text{foo} X_{\gamma}(k_i(t_1, \ldots, t_n)) \)
- All SLD proofs over the Horn Clauses in \( \mathbb{P} \).
Observe that the predicate \( \text{foo}(X) \) means \( X_{\text{foo}} \), which in the category \( \mathbb{C} \) is \( (\text{foo} \circ id_{\text{foo}}) \). It is easily shown \( \text{foo} \) extends to an endofunctor on \( \mathbb{C} \) (the case where \( \text{foo} \) is a monad is studied in [18]).

In the case of e.g., the list datatype, the arrow \( X_{\text{E}_1}(\gamma) \xrightarrow{\text{cons}} \text{foo} X \), becomes \( X_{\gamma} \times \text{list}(\gamma) \xrightarrow{\text{cons}} \text{list}(X_{\gamma}) \) and \( X_{\text{E}_1}(\gamma)\langle t_1, \ldots, t_n \rangle \xrightarrow{id} \text{foo}(X_{\gamma})(k(t_1, \ldots, t_n)) \) becomes \( X_{\gamma} \times \text{list}(\gamma)\langle t_1, t_2 \rangle \xrightarrow{\text{cons}} \text{list}(X_{\gamma})(\text{cons}(t_1, t_2)) \) which corresponds to the proof rule

\[
\text{list}(X_{\gamma})(\text{cons}(t_1, t_2)) \xrightarrow{id} X_{\gamma}(t_1), \text{list}(X_{\gamma})(t_2)
\]

\( \mathbb{C} \) is in fact the least finite product category satisfying the constraints in \( \mathbb{C} \) in which non-deterministic SLD proofs and the datatype proof rules

\[
X_{\text{E}_1}(\gamma)\langle t_1, \ldots, t_n \rangle \xrightarrow{id} \text{foo} X_{\gamma}(k(t_1, \ldots, t_n))
\]

of the preceding theorem, occur as arrows (with directions reversed). The proof makes use of the universal mapping properties discussed in the preceding sections, and is beyond the scope of this paper. We refer the reader to [18] for details.

### 4.1 Encapsulation

Finally we show how to lift a datatype definition with the syntax

```plaintext
datatype 'a foo = c0 | c1 ... | k1 of E_1('a) | ... | lim 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 \( \mathbb{C} \), possibly defined by induction in terms of the initial algebra \( \text{foo}(\gamma) \) for a sort \( \gamma \) occurring in the program \( P \).

All we need to change is the definition of \( J \), which now must be a disjoint union of the \( J_{\gamma} \) and the diagram

\[
\alpha \times \alpha \xrightarrow{id_{\alpha} \times f} \alpha \times \rho.
\]

The generic predicate \( m_j \equiv X_j \xrightarrow{\text{pred}[f]} (\alpha \times \rho \ll id_{\alpha \times \rho} \gg) \) resulting from this extra component of \( J \) is called \( \text{pred}[f] \) in the program syntax. It will automatically satisfy the condition that \( \text{pred}[f] \chi_{\alpha \times \rho}(\text{OO}) \) is an isomorphism in the category \( \mathbb{C} \). That is to say, the construction “hard-wires” the graph of the function \( f \) into the binary relation \( \text{pred}[f] \). When interpreted in any model \( \text{pred}[f] \chi_{\alpha \times \rho}(\text{OO}) \) will be true, and in any resolution proof in \( \mathbb{C} \) this goal is isomorphic to the empty goal of its sort. We illustrate with an example based on the code in the example in subsection (1.1). 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 $\text{list}(\gamma) \xrightarrow{\text{length}} \text{int}$ in $\mathbb{J}$ (and in $\mathbb{C}$), and to the arrow $\text{list}(\gamma) \xrightarrow{\langle \text{id}, \text{length} \rangle} \text{list}(\gamma) \times \text{int}$ in $\mathbb{J}$, mapped by $\sigma$ to itself in $\mathbb{C}_{P[\sigma]}$. The objects in category $\mathbb{C}_{P[\sigma]}$ are pairs $(A, S)$ where $S$ must satisfy the requirement:

for every arrow $A \xrightarrow{\varphi} \text{list}(\gamma)$, the composition $\varphi \circ \langle \text{id}, \text{length} \rangle$ is in $S(\text{list}(\gamma) \times \text{int})$.

This is guaranteed by the condition that $S$ is a subfunctor of $\mathbb{C} \langle A, \sigma(\omega) \rangle$. The predicate $\text{pred[\text{length}]}$ is represented in $\mathbb{C}_{P[\sigma]}$ as the generic

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

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

$$(\text{list}(\gamma), \langle \langle \text{id}, \text{length} \rangle \rangle) \xrightarrow{\text{id}} (\text{list}(\gamma), \langle \emptyset \rangle)$$

which has the inverse $(\text{list}(\gamma), \langle \langle \text{id}, \text{length} \rangle \rangle) \xleftarrow{\text{id}} (\text{list}(\gamma), \langle \emptyset \rangle)$ since, perhaps surprisingly at first, $\langle \langle \text{id}, \text{length} \rangle \rangle \subseteq \langle \emptyset \rangle$. This holds because condition (4.1) above now reads

for every arrow $\text{list}(\gamma) \xrightarrow{\varphi} \text{list}(\gamma)$, the composition $\varphi \circ \langle \text{id}, \text{length} \rangle$ is in $S(\text{list}(\gamma) \times \text{int})$.

for every $S$, including $\langle \emptyset \rangle$. Now take $\varphi$ to be the identity. Thus

$$\text{pred[\text{list}]}(x, \text{length}(x))$$

is represented by an isomorphism onto its sort in $\mathbb{C}_{P[\sigma]}$.

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 integrate constraints on terms and uniform proofs [9, 6] and monads [18]. We hope it will prove a natural vehicle for declarative approach to the limited
amounts of control and state that are really 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 $C$, when $C$ is suitably chosen (for example with recursively enumerable pullbacks). It should be investigated.

References