Categorical Bottom-Up Semantics for Logic Programming

Ed Morehouse

Wesleyan University

Outline

The Dream of Programming in Logic

Why try to program in logic?

- Logic is declarative: say *what* you want, not *how* to get it
- Transfer work from people to machines
- Reduce error
- Reduce the "semantic gap" between specifications and programs

To a limited extent, the dream has been realized with definite clause logic.

The Dream Deferred

Then why are we not satisfied?

- Systems lack many basic programming features (e.g. modules, abstract datatypes, higher-order functions). Even arithmetic can be cumbersome.
- Some features get bolted on but are often not consistent with the the underlying logical theory.
- Still a large semantic gap (sometimes made worse by the shortcomings of the systems).

What can we do?

- Extend the logic
- Add non-logical features in a principled way that doesn't destroy declarative transparency of programs

The Dream Deferred

Then why are we not satisfied?

- Systems lack many basic programming features (e.g. modules, abstract datatypes, higher-order functions). Even arithmetic can be cumbersome.
- Some features get bolted on but are often not consistent with the the underlying logical theory.
- Still a large semantic gap (sometimes made worse by the shortcomings of the systems).

What can we do?

- Extend the logic
- Add non-logical features in a principled way that doesn't destroy declarative transparency of programs

2 definite clause logic programming in SET

- Definite Clause Logic
- Herbrand Models
- Immediate Consequences
- Answers and Unification
- SLD Resolution

Definite Programs and Goals

definite goal and program clauses Let $\mathcal{L}$ be a first-order language. We define the set of **goal formulas** $\mathcal{G}$ and **program formulas** $\mathcal{D}$ (also called definite clauses) by the following grammar over atoms, A:

$$G ::= \top \mid A \mid G \wedge G$$
$$D ::= A \leftarrow G$$

definite program A finite $P \subset \mathcal{D}$

The World According to Herbrand

Herbrand universe ($U_{\mathcal{L}}$) For language $\mathcal{L}$, the set of all closed *terms* that can be formed from the constant and function symbols of $\mathcal{L}$.

Herbrand base ($B_{\mathcal{L}}$) For language $\mathcal{L}$, the set of all closed *atomic formulas* (atoms) that can be formed from the predicate symbols of $\mathcal{L}$ with elements of the Herbrand universe as arguments.

We can relativize these a set of $\mathcal{L}$-formulas P, by writing U_P and B_P

Herbrand Interpretations and Models

Herbrand interpretation Any subset of the Herbrand base (representing the closed atoms that are true in the interpretation). The set of all such is written $2^{B_{\mathcal{L}}}$.

Herbrand model for a set of closed $\mathcal{L}$-formulas T, a Herbrand interpretation of $\mathcal{L}$ that is a model for T.

Definite Clauses and Herbrand Models

proposition If a set of *definite clauses* is satisfiable then it has a Herbrand model.

model intersection property If P is a definite program and $\{M_i\}_{i:I}$ a non-empty set of Herbrand models for P, then $\bigcap_{i:I} M_i$ is also a Herbrand model for P.

least Herbrand model (M_P) The intersection of all Herbrand models for P. (this always exists since $B_P \models P$ so the set of such is nonempty)

theorem (Kowalski & van Emden 1976)

$$M_P = \{A : B_P \mid P \models A\}$$

The T_P Operator on Herbrand Interpretations

immediate consequences operator (T_P) For definite program P, the function on Herbrand interpretations

$$T_P : 2^{B_P} \to 2^{B_P}$$

defined by

$$I \longmapsto \{A : B_P \mid (A \leftarrow A_1 \wedge \cdots \wedge A_n) \text{ is a ground instance of of a clause in } P \text{ and } \{A_1, ..., A_n\} \subset I\}$$

Least Herbrand Model as a Fixpoint

proposition For Herbrand interpretation I of definite program P,

$$I \models P \iff T_P(I) \subset I$$

fixed point characterization of least Herbrand model For definite program P,

$$\begin{aligned} M_P &= \text{lfp}\ (T_P) \\ &= T_P \uparrow (\emptyset) \\ &= T_P(\emptyset) \cup T_P(T_P(\emptyset)) \cup \cdots \end{aligned}$$

Correct Answers

correct answer For definite program P and definite goal G, a substitution θ such that $P \models \forall(G\,\theta)$

proposition For definite program P, definite goal G, and substitution θ such that $G\,\theta$ is ground, θ is a correct answer iff $M_P \models G\,\theta$.

Unification

unifying substitution (unifier) For expressions E and F, a substitution θ such that $E\,\theta = F\,\theta$.

most general unifier (mgu) Unifier θ is most general if for every unifier φ, there is a substitution σ such that $\varphi = \theta \cdot \sigma$

unification theorem Unification in first order logic is decidable. If a pair of terms are unifiable, then the unification algorithm computes their mgu, else it reports failure.

SLD Derivation

SLD step ($\rightsquigarrow$) For definite program clause $P = A \leftarrow B_1 \wedge \cdots \wedge B_m$, definite goal clause $G = A_1 \wedge \cdots \wedge A_i \wedge \cdots \wedge A_n$ and substitution $\theta = mgu(A_i, A)$,

$$G \underset{P,A_i}{\overset{\theta}{\rightsquigarrow}} G'$$

where

$$G' = (A_1 \wedge \cdots \wedge A_{i-1} \wedge \overbrace{B_1 \wedge \cdots \wedge B_m}^{\text{tail of P replaces } A_i} \wedge A_{i+1} \wedge \cdots \wedge A_n)\ \theta$$

this is essentially tracing the implications, "$\leftarrow$" from the program backwards.
i.e. "$G\ \theta$ would be true if $G'\ \theta$ were true"

SLD Derivation

SLD derivation For definite program P and goal G, a (possibly infinite) chain of derivation steps starting from $G = G_0$:

$$G_0 \overset{\theta_0}{\underset{P_0,A_0}{\rightsquigarrow}} G_1 \overset{\theta_1}{\underset{P_1,A_1}{\rightsquigarrow}} \cdots$$

where A_i is an atom in G_i and P_i is a clause in P (with variables renamed to avoid clashes).

SLD proof A finite SLD derivation ending in $\top$ (success).

computed answer The substitution that is the composition of mgus in a proof. (this is what is returned by a PROLOG interpreter)

Soundness and Completeness of SLD Derivation

soundness of SLD resolution (Clark 1979) For definite program P and definite goal G, every computed answer for $(P \vdash_{SLD} G)$ is a correct answer.

completeness of SLD resolution (Clark 1979) For definite program P and definite goal G, for every correct answer φ for (P, G) there exists a computed answer θ and substitution γ such that $\varphi = \theta \cdot \gamma$.

3 categorical syntax, derivation and semantics for definite clause logic

- categorical syntax
- categorical SLD derivation
- categorical semantics

Categorical Signature

a triple, $\Sigma = (\mathbb{C}, \mathcal{D}, \vec{B})$ where:

- $\mathbb{C}$ is a category with finite products and distinguished limits
  The objects of $\mathbb{C}$ are called **sorts**
- $\mathcal{D} \subset \mathbb{C}^{\rightarrow}$
  whose elements are called **terms**
- $\vec{B} = B_1, ..., B_n$ is a sequence of distinct monics in $\mathbb{C}$ with the property that their pullbacks along coterminal arrows in $\mathcal{D}$ exist.
  We call these **predicate tokens**

Interpretation of a Language in a Signature

For sorted first-order language $\mathcal{L}$ and signature $\Sigma = (\mathbb{C}, \mathcal{D}, \vec{B})$, a **syntactic interpretation** of $\mathcal{L}$ into Σ is a function $M : \mathcal{L} \to \mathbb{C}$ such that:

sort	σ	$\longmapsto$	$M(\sigma)$ where $\mathrm{id}_\sigma \in \mathcal{D}$	
constant symbol	$c : \sigma$	$\longmapsto$	$M(c) : \mathbf{1} \to M(\sigma)$	
function symbol	$f : (\rho \to \sigma)$	$\longmapsto$	$M(f) : (M(\rho) \to M(\sigma))$	$\in \mathcal{D}$
relation symbol	$R : \sigma$	$\longmapsto$	$M(R) : (\rho \rightarrowtail M(\sigma))$	$\in \vec{B}$

such that the image of every term in $\mathcal{L}$ (inductively generated by the function symbols) is a term in $\mathbb{C}$ – i.e. in $\mathcal{D}$.

Syntactic Interpretation of Variables

Categorical syntax is inherently variable-free, but to keep track of which variables in the language are under consideration, we use the following convention:
For $\vec{x}$ a sequence of free $\mathcal{L}$-variables of sort $\vec{\rho}$:

$$M(\vec{x}) = M(\rho_1) \times ... \times M(\rho_n) : \mathbb{C}$$

Syntactic Interpretation of Terms

For term t of (out) sort ρ with free variables among $\vec{x}$:

$$M_{\vec{x}}(t) : \mathbb{C}(M(\vec{x}) \to M(\rho))$$

is defined:

variable $M_{\vec{x}}(x_i) = \pi_i$, a canonical projection

constant symbol $M_{\vec{x}}(c) = \ !_{M(\vec{x})} \cdot M(c)$

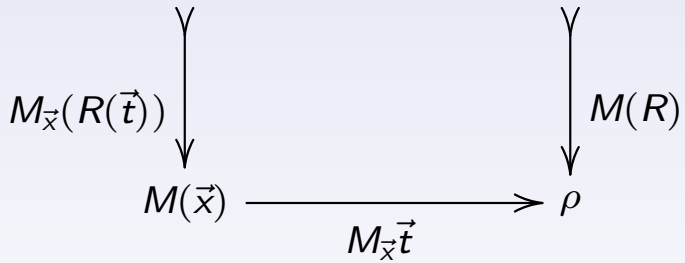
function symbol applied to terms $M_{\vec{x}}(f(\vec{t})) = \langle M_{\vec{x}}(t_1), ..., M_{\vec{x}}(t_n) \rangle \cdot M(f)$

Syntactic Interpretation of Atomic Formulas

For predicate token R of sort ρ and terms $\vec{t}$ with their free variables among $\vec{x}$

$$M_{\vec{x}}(R(\vec{t}))$$

is the monic targeted at $M(\vec{x})$ that is the *pullback* of R along $M_{\vec{x}}\vec{t}$ $\quad (= \langle M_{\vec{x}}(t_1), ..., M_{\vec{x}}(t_n) \rangle)$



(recall that the pullback of a monic is monic)

Definite Clause Diagrams

For a definite program or goal clause C with its free variables among $\vec{x}$, the set of monics interpreting its atomic formulas, each targeted at sort $M(\vec{x})$. For program clause diagrams, we also annotate which monic interprets the head of the clause.

note For definite clause logic, we can get away with such a simple notion of formula diagram because goals are just conjunctions of atoms.

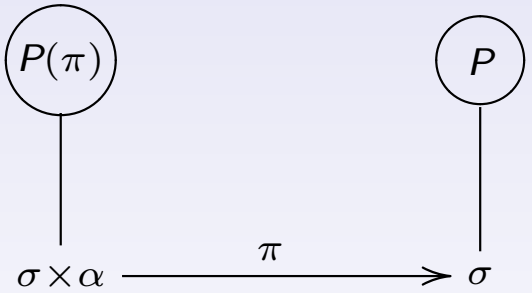
The empty diagram corresponds to goal "$\top$".

Sort Extension

Just as we can add "dummy variables" to a formula of first order logic:

$$\varphi(x) \longmapsto \varphi(x, y)$$

we can extend the sort of a diagram by pulling it back along a projection:



This allows us to bring any set of diagrams over a common sort, namely that of a product of their individual sorts.

Categorical Substitution

Let $\theta = \{\vec{x} := \vec{t}\}$ be an $\mathcal{L}$-substitution and $\vec{y}$ be a sequence of distinct variables containing all free variables in $\vec{t}$. Then the categorical substitution $\Theta_{\vec{y}}$ is the morphism:

$$\Theta_{\vec{y}} = \langle M_{\vec{y}}(t_1), ..., M_{\vec{y}}(t_n) \rangle : M(\vec{y}) \longrightarrow M(\vec{x})$$

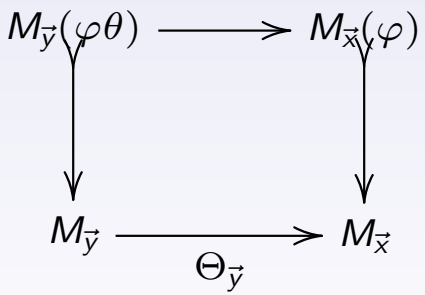
- The contravariance is due to the fact that substitution acts on inputs (the free variables) rather than outputs.
- since they're just arrows, substitutions act on terms and formulas the same way other arrows do.

Categorical Substitution

applying substitutions to terms For $\mathcal{L}$-term s with free variables among $\vec{x}$ and substitution θ as above:

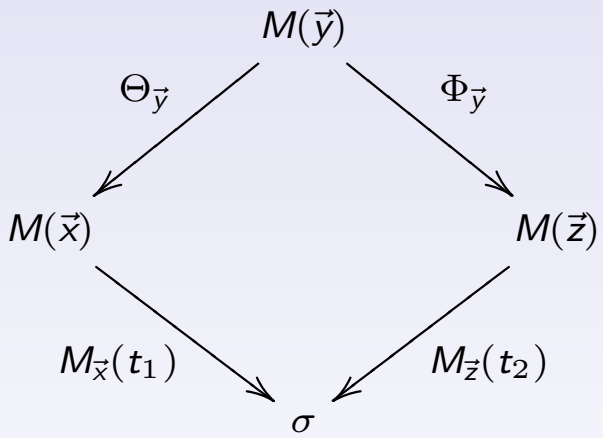
$$M_{\vec{y}}(s\,\theta) = \Theta_{\vec{y}} \cdot M_{\vec{x}}(s)$$

applying substitutions to atomic formulas For atomic formula φ, $M_{\vec{y}}(\varphi\theta) = (\Theta_{\vec{y}})^{\#}(M_{\vec{x}}(\varphi))$:



Categorical Unification

unification of terms terms t_1, t_2 are unifiable if there exist categorical substitutions $\Theta_{\vec{y}}, \Phi_{\vec{y}}$ such that



Categorical Unification

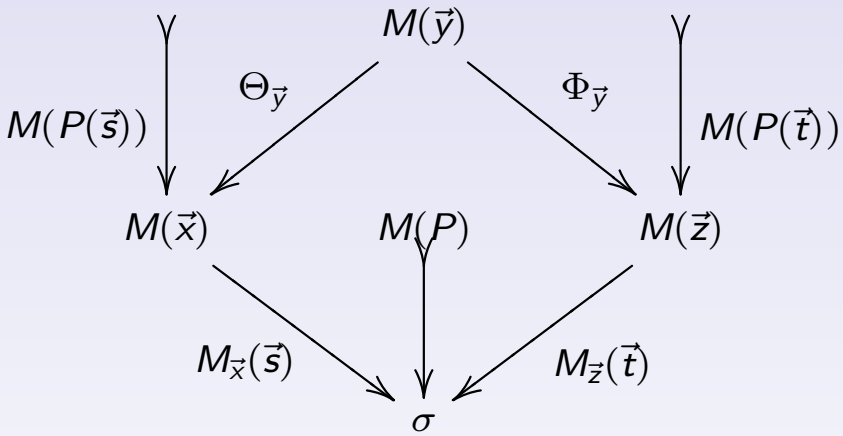
unification of atomic formulas atomic formulas $P_1(\vec{s}), P_2(\vec{t})$ are unifiable if

- the interpretations of their predicate symbols are identical
- the interpretations of their argument terms are unifiable

most general unifiers in either case, a unifier is most general if the square is a pullback.

Categorical Unification

unification of atomic formulas atomic formulas $P_1(\vec{s}), P_2(\vec{t})$ are unifiable if



most general unifiers in either case, a unifier is most general if the square is a pullback.

Categorical SLD Derivation Step

For goal clause diagram $M_{\vec{x}}(G = A_1 \wedge \cdots \wedge A_n)$ and program clause diagram $M_{\vec{z}}(C = A \leftarrow B_1 \wedge \cdots \wedge B_m)$, goal clause diagram $M_{\vec{y}}(G')$ is derived from $M_{\vec{x}}(G)$ and $M_{\vec{z}}(C)$ by $(\Theta_{\vec{y}}, \Phi_{\vec{y}})$ if

- $M_{\vec{x}}(A_i)$ is the selected atom of $M_{\vec{x}}(G)$
- $(\Theta_{\vec{y}}, \Phi_{\vec{y}})$ is the mgu of $M_{\vec{x}}(A_i)$ and $M_{\vec{z}}(A)$

in which case,

$$\begin{array}{rlll} M_{\vec{y}}(G') = & \Theta_{\vec{y}}{}^{\sharp}\ (M_{\vec{x}}(A_1) & \cup \cdots \cup & M_{\vec{x}}(A_{i-1})) \\ \cup & \Phi_{\vec{y}}{}^{\sharp}\ (M_{\vec{z}}(B_1) & \cup \cdots \cup & M_{\vec{z}}(B_m)) \\ \cup & \Theta_{\vec{y}}{}^{\sharp}\ (M_{\vec{x}}(A_{i+1}) & \cup \cdots \cup & M_{\vec{x}}(A_n)) \end{array}$$

where $\Theta_{\vec{y}}$ is the **goal substitution** and $\Phi_{\vec{y}}$ the **program substitution**.

Categorical SLD Derivations and Answers

- We write $M_{\vec{x}}(G) \overset{\Theta_{\vec{y}}, \Phi_{\vec{y}}}{\underset{A_i, C}{\rightsquigarrow}} M_{\vec{y}}(G')$ for a derivation step.

  (we suppress the program substitution if it is not relevant)
- By chaining these steps, we get a **categorical SDL derivation**.
- If the derivation is finite and the last goal is the empty goal diagram over its sort, we have a **categorical SLD proof**.
- The **computed answer** of a proof is the composition of the *goal substitutions*

$$\Theta_n \cdot \Theta_{n-1} \cdot \ldots \cdot \Theta_1$$

  (note the reversal of order, corresponding to the contravariance of categorical substitution)

Categorical Semantics

- Once we are able to translate a language into a category that we will call **syntactic**, we want to be able to do model theory in the context of categories as well.
- Structure-preserving (in this case, limit-preserving) functors allow us to interpret one logic programming category in another.
- We call the target of a categorical interpretation the **semantic** category.

4 Yoneda semantics for a category of generic predicates

- syntactic categories with generic predicates
- interpretation in the Yoneda topos
- consequences

Generic Predicates

In pure logic programming, we want to begin with a syntactic category that is as free as possible, and for the logical structure to be supplied (entirely) by a program.

generic predicate We say that monic X is generic with respect to a collection of arrows, $\mathcal{D}$ if:

- $\forall t \in \mathcal{D}$ coterminal with X the pullback $t^{\#}(X)$ exists
  (so predicates can be instantiated at all sort-correct terms)
- no such pullback is an isomorphism
  (so no instance of a predicate is true *in the syntax*)

There is a construction that allows us to add generic predicates to a category, of which we will consider a simple special case.

A Category With a Generic Predicate ($\mathbb{C}[X]$)

For $\mathbb{C}$ a category with finite products and $b : \mathbb{C}$ (the sort at which we want to add a generic predicate), we construct the category $\mathbb{C}[X]$ to have:

objects pairs $\langle A, S \rangle$ where $A : \mathbb{C}$ and $S \subset \mathbb{C}(A \to b)$ is finite

arrows triples $\langle A, S \rangle \xrightarrow{f} \langle B, T \rangle$ where $f : \mathbb{C}(A \to B)$ such that

$$f \cdot T \subset S$$

we call f the *label* of this arrow

We claim that this category will embed an isomorphic copy of $\mathbb{C}$, and contain a new generic predicate of sort b.

Properties of $\mathbb{C}[X]$

This construction preserves the categorical properties we want our syntax to have:

- $\mathbb{C}[X]$ has finite products:
  - terminal object $\langle \mathbf{1}, \emptyset \rangle$
  - binary products $\langle A, S \rangle \times \langle B, T \rangle = \langle A \times B \ , \ \pi_{1_\mathbb{C}} \cdot S \cup \pi_{2_\mathbb{C}} \cdot T \rangle$
- $\mathbb{C}[X]$ inherits canonical structure from $\mathbb{C}$
- The **inclusion functor** $\iota : \mathbb{C} \to \mathbb{C}[X]$, mapping:

$$\begin{array}{rcl} A & \longmapsto & \langle A, \emptyset \rangle \\ f : A \to B & \longmapsto & \langle A, \emptyset \rangle \xrightarrow{f} \langle B, \emptyset \rangle \end{array}$$

is a continuous full and faithful embedding.

Special Subobjects

indeterminate subobject (X) In $\mathbb{C}[X]$, the *indeterminate subobject* X of sort b is the (monic representing its) subobject:

$$\langle b, \{\mathrm{id}_b\}\rangle \overset{id_b}{\rightarrowtail} \langle b, \emptyset\rangle$$

atomic goal the pullback of the X along the ι-image of an arrow $t : \mathbb{C}(A \to b)$:

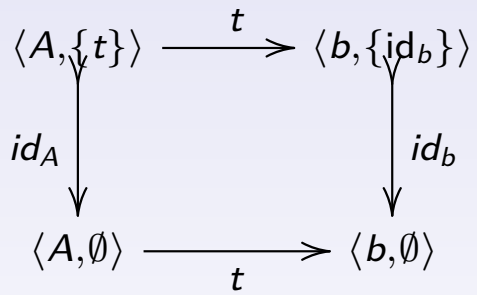
$$\langle A, \{t\}\rangle \overset{\mathrm{id}_A}{\rightarrowtail} \langle A, \emptyset\rangle$$

goal a monic labeled by the identity: $\langle A, S\rangle \overset{\mathrm{id}_A}{\rightarrowtail} \langle A, \emptyset\rangle$

The Indeterminate Subobject is Generic

proposition Indeterminate subobject X of sort b is a generic predicate with respect to the image of arrows of $\mathbb{C}$ under the inclusion map $\iota : (\mathbb{C} \rightarrow \mathbb{C}[X])$

proof For any $t : \mathbb{C}(A \rightarrow b)$



The left arrow is the pullback of X along $\iota(t)$.
It will never be an isomorphism since $t \notin \emptyset$.

Syntactic Categories of Generic Predicates

- We can either iterate or generalize this construction to add as many generic predicates (of whatever sorts) as we need for the relation symbols of a language.
- We will call such generic predicate categories $\mathbb{C}[\vec{X}]$, where $\vec{X}$ is a sequence of the predicate symbols we are interested in.

Interpretation of $\mathbb{C}[\vec{X}]$ in the Yoneda Topos

A continuous functor

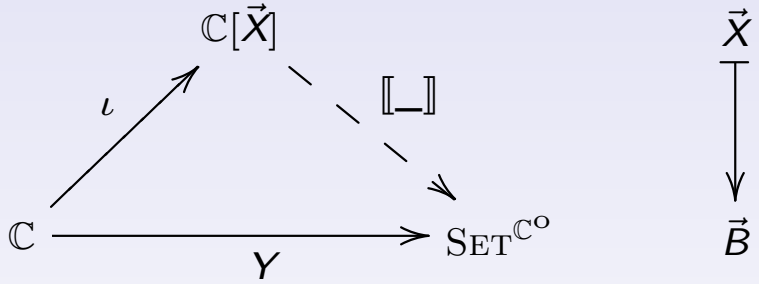
$$[\![\_]\!] : \mathbb{C}[\vec{X}] \to \mathrm{SET}^{\mathbb{C}^{\mathrm{o}}}$$

- agreeing with the Yoneda embedding on $\iota(\mathbb{C})$
- mapping the source of X_i to some canonical subobject (i.e. pointwise subset) of $\mathbb{C}(\_ \to b_i)$
  (i.e. $[\![\_]\!](\Box X_i)(A) \subset \mathbb{C}(A \to b_i)$)

Interpretation of $\mathbb{C}[\vec{X}]$ in the Yoneda Topos

A continuous functor

$$[\![\_]\!] : \mathbb{C}[\vec{X}] \to \mathrm{SET}^{\mathbb{C}^{\mathrm{o}}}$$



a universal mapping property tells us that this is enough to uniquely determine $[\![\_]\!]$

Category of Interpretations

- Let $\mathcal{I}^{\mathbb{C}}_{\vec{X}}$ be the set of interpretations of $\mathbb{C}[\vec{X}]$ in $\mathrm{SET}^{\mathbb{C}^{\mathrm{o}}}$.
- We can make $\mathcal{I}^{\mathbb{C}}_{\vec{X}}$ into a poset, $(\mathcal{I}^{\mathbb{C}}_{\vec{X}}, \sqsubset)$ where

$$[\![\_]\!] \sqsubset [\![\_]\!]' \iff \forall (X_i \in \vec{X}).\ [\![X_i]\!] \subset [\![X_i]\!]' \text{pointwise}$$

- This poset $(\mathcal{I}^{\mathbb{C}}_{\vec{X}}, \sqsubset)$ is a complete lattice, with meet and join given pointwise and top and bottom elements:

$$[\![X_i]\!]^{\top} = \mathbb{C}(\_, b_i) \xrightarrow{id} \mathbb{C}(\_, b_i)$$
$$[\![X_i]\!]^{\bot} = \mathbf{0} \xrightarrow{\mathrm{i}} \mathbb{C}(\_, b_i) \qquad (\mathbf{0} = \lambda x.\ \emptyset \text{ is initial})$$

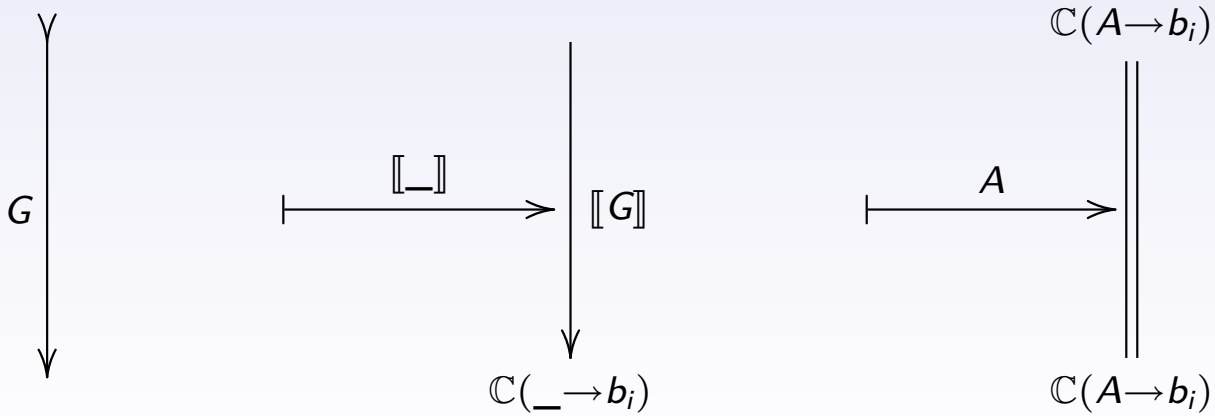
- Let $\mathrm{INT}^{\mathbb{C}}_{\vec{X}}$ be this poset, as a category.

Models and Truth

model An interpretation $[\![\_]\!] : \mathrm{INT}^{\mathbb{C}}_{\vec{X}}$ is a *model* of a definite clause program P if for every clause $(hd \leftarrow tl) \in P$, we have $[\![tl]\!] \subset [\![hd]\!]$, pointwise in $\mathrm{SET}^{\mathbb{C}^o}$.

truth of a goal in an interpretation A goal G is true in interpretation $[\![\_]\!]$ if its image is an isomorphism.

$\mathbb{C}[\vec{X}]$ $\quad$ $\mathrm{SET}^{\mathbb{C}^o}$ $\quad$ SET



Soundness

of SLD resolution with respect to models of a program

theorem

If interpretation $[\![\_]\!] : \mathbb{C}[\vec{X}] \to \mathrm{SET}^{\mathbb{C}^{\mathrm{o}}}$ is a model for program P and G is a goal, if there is an categorical SLD derivation $G \rightsquigarrow \overset{\Theta}{\cdots} \rightsquigarrow \top$ with computed answer substitution Θ, then $\Theta^{\#} G$ is true in $[\![\_]\!]$.

Immediate Consequences (E_P)

E_P is the categorical analog to T_P of Kowalski-van Emden semantics.

$$E_P : \mathrm{INT}^{\mathbb{C}}_{\vec{X}} \to \mathrm{INT}^{\mathbb{C}}_{\vec{X}}$$

- E_P is a continuous operator on $\mathcal{I}^{\mathbb{C}}_{\vec{X}}$ and thus has a least fixed point, $[\![\_]\!]^*$, which is the categorical analogue of the least Herbrand model.
- $[\![\_]\!]^*$ allows us to prove a completeness theorem for categorical SLD resolution as well.

5 beyond definite clause logic

Toward extensions

- so far, we have developed a categorical syntax, derivation system and bottom-up semantic only for definite clause logic, which, as we pointed out in the beginning lacks desirable features
- some of these can be added by changing the syntactic category
- some can be added by passing to indexed categories

Syntactic Categories with more Structure

Consider a simple predicate for the factorial relation

$$\texttt{fact}(0, 1).$$
$$\texttt{fact}(\texttt{X} + 1, (\texttt{X} + 1) * \texttt{Y}) : -\texttt{fact}(\texttt{X}, \texttt{Y}).$$

In a syntactic category in which fact is generic (and in PROLOG), this will not do what we want.
However, we could imagine this as a program for a syntactic category *modulo arithmetic*.
Then, for example, the term $2 + 4$ and the term $3 * 2$ would be *the same arrow in the syntax*.

(strict) Indexed Categories

indexed category For category $\mathbb{C}$, a functor $P : \mathbb{C}^{\mathrm{o}} \to \mathrm{CAT}$

For $\sigma : \mathbb{C}$, $P(\sigma)$ is called the *fiber over* σ
and for $f :: \mathbb{C}$, $P(f)$ is called a *reindexing functor*.

logic programming in indexed categories

- The base category $\mathbb{C}$ becomes the "state" we wish to index over. In the simplest case, this is the category of sorts and terms.
- Predicates are objects in the fibers over their state.
- The action of the reindexing functors and the structure of the fibers together with the program generate the resolution steps.

Hereditarily Harrolp Logic

- Hereditarily Harrop logic extends clausal logic by allowing implication and in goals:

$$G ::= \top \mid A \mid G \wedge G \mid G \vee G \mid D \rightarrow G \mid \exists_{x:\alpha} G \mid \forall_{x:\alpha} G$$
$$D ::= A \mid A \Leftarrow G \mid D \wedge D \mid \forall_{x:\alpha} D$$

- In [4] it was shown that this logic fragment can be made into a logic programming language with a proof search system called **uniform proof**, that for every non-atomic formula, applies the natural deduction proof rule that introduces its primary connective or quantifier.
- indexing over programs, allows us to handle implication in goals ([2])
- indexing over freely-adjoined constants gets us universal quantification in goals as well (Krishnan)

references

S. Finkelstein, P. Freyd, and J. Lipton.

Logic programming in tau categories.

Lecture Notes in Computer Science, 933, 1995.

S. Finkelstein, P. Freyd, and J. Lipton.

A new framework for declarative programming.

Theoretical Computer Science, 300:91–160, 2003.

J. W. Lloyd.

Foundations of Logic Programming.

Springer-Verlag, second, extended edition, 1993.

D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov.

Uniform proofs as a foundation for logic programming.

Annals of Pure and Applied Logic, 51, 1991.