Predicate Calculus for Algebraic Theories

Christian Williams
williams@math.ucr.edu

University of California, Riverside

Chapman
Feb. 21, 2020

Contents

Motivation RChain

RChain is a public computing infrastructure based on a concurrent language called the reflective higher-order π calculus, or ρ calculus.

π calculus

communication unifies internal and external.

$$\bar{n}y \mid n(x).p \Rightarrow p[y/x]$$

ρ calculus

reflection unifies data and code.

$$@ : \mathrm{P} \rightleftharpoons \mathrm{N} : *$$

$$\mathrm{out}(n, q) \mid \mathrm{in}(n, x.p) \Rightarrow p[@q/x]$$

Motivation RChain

In the ρ calculus, names have *form*, which provides large-scale structure to a network. We aim to use this structure, to think globally about the web.

Namespace Logic [1] is a framework for this reasoning. The theory is

$$\rho \text{ calculus} + \text{ first-order logic } + \text{recursion} = \mathrm{NL}.$$

A formula φ of NL expresses a property of terms in the language. This is a type, used to condition the inputs and outputs of programs.

Motivation RChain

Example

If φ is the "namespace" that a community trusts, they may use a "firewall" which asks any process: who else do you know?

$$\texttt{soleAccess}_\varphi := \texttt{in}(\varphi, \top) \wedge \neg[\texttt{in}(\neg\varphi, \top)].$$

Security and liveness properties of systems are one of many applications of namespace logic. It can also function as a *query language*, which allows one to search, not by keywords, but by the actual *structure* of code.

Motivation RChain

How do we construct this logic *generally*, for any language?

$$\begin{aligned} \text{Language} &: \text{algebraic theory} \quad \mathbb{T} \\ \text{Logic} &: \text{presheaf topos} \quad \hat{\mathbb{T}}. \end{aligned}$$

Algebraic Theories

A class of algebraic structure can be specified by a category in which

objects	:	products of base types	$\mathtt{t}$	$\in \mathcal{T}$
morphisms	:	composites of operations	$\mathtt{op}$	$\in \mathcal{O}$
diagrams	:	equations	$(\mathtt{op}_1, \mathtt{op}_2)$	$\in \mathcal{E}$.

This provides a presentation by generators and relations; the free cartesian category is the "theory" of this structure.

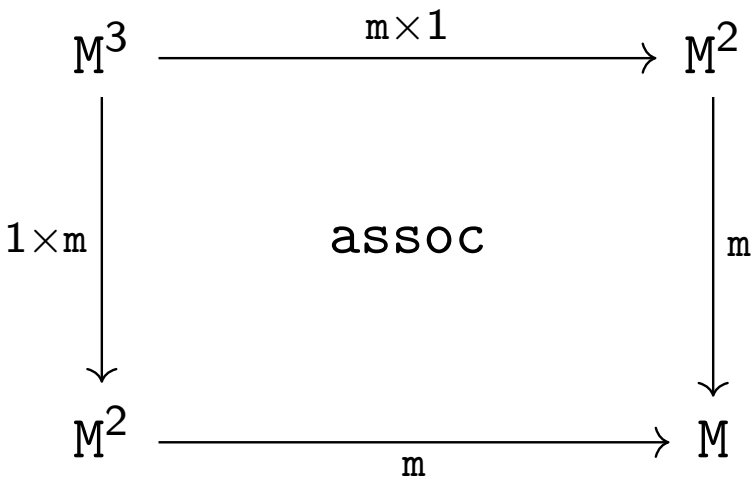
Definition

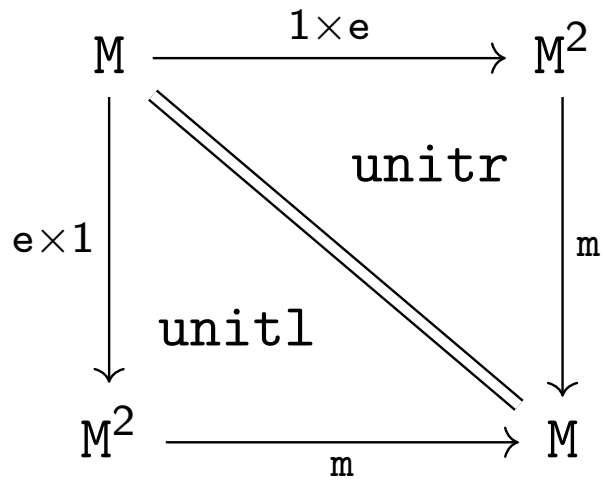
An **algebraic theory** is a category $\mathtt{T}$ equipped with presentation $(\mathcal{T}, \mathcal{O}, \mathcal{E})$ which exhibits $\mathtt{T}$ as the free cartesian category on the presentation.

Algebraic Theories

Th.Mon

$$\mathtt{m} : \mathtt{M}^2 \to \mathtt{M} \leftarrow \mathtt{1} : \mathtt{e}$$





Presheaf Toposes

A **presheaf** on a category $\mathtt{T}$ is a functor $\mathtt{T}^{\mathrm{op}} \to \mathrm{Set}$. There is a natural way to embed $\mathtt{T}$ into its *category of presheaves*.

Definition

The **Yoneda embedding** of $\mathtt{T}$ is a functor

$$y_{\mathtt{T}} : \mathtt{T} \to [\mathtt{T}^{\mathrm{op}}, \mathrm{Set}] =: \hat{\mathtt{T}}$$

which is defined

$$\begin{aligned} \mathtt{s} &\mapsto \mathtt{T}(-, \mathtt{s}) &&=: \hat{\mathtt{s}} \\ \mathtt{o}\downarrow \quad &\downarrow \mathtt{T}(-, \mathtt{o}) &&=: \hat{\mathtt{o}} \\ \mathtt{t} &\mapsto \mathtt{T}(-, \mathtt{t}) &&=: \hat{\mathtt{t}}. \end{aligned}$$

Presheaf Toposes

Definition

A **topos** is a category $\mathcal{F}$ such that

- $\mathcal{F}$ is cartesian closed,
- $\mathcal{F}$ has finite co/limits, and
- $\mathcal{F}$ has a subobject classifier Ω.

Any presheaf category $\hat{\mathbb{T}}$ is a topos, with a rich internal logic. In particular, we can interpret and utilize subobjects as *predicates*.

Lemma

For any presheaf $P : \mathbb{T}^{\mathrm{op}} \to \mathrm{Set}$, *subfunctors of* P *form a Heyting algebra*

$$\hat{\mathbb{T}}[P, \Omega] =: \mathrm{Sub}(P).$$

Presheaf Toposes

Let $\mathtt{T}$ be a category and $\mathtt{t}$ be an object. A **sieve** on $\mathtt{t}$ is a class of morphisms $S \subset \mathrm{Mor}(\mathtt{T})$ which is closed under precomposition.

Lemma

There is a natural bijection of sieves on $\mathtt{t}$ *and subfunctors of* $\mathtt{T}(-, \mathtt{t})$.

Puzzle: what is a sieve in an algebraic theory $\mathtt{T}$?

The Predicate Calculus

The operations of $\mathtt{T}$ generate sieves:

$$S(\mathtt{m}) \ \ldots \ \rightrightarrows \ \mathtt{M}^2 \xrightarrow{\ \mathtt{m}\ } \mathtt{M} \xleftarrow{\ \mathtt{e}\ } 1 \ \leftleftarrows \ \ldots \ S(\mathtt{e}).$$

These act as basic building blocks for predicates; they correspond to subfunctors $\check{\mathtt{m}}, \check{\mathtt{e}} \rightarrowtail \mathtt{T}(-, \mathtt{M})$, to which we can apply the operations of the Heyting algebra. We can then build formulae such as:

$$[\check{\mathtt{e}}] \vee [\check{\mathtt{m}}].$$

The Predicate Calculus

To complete the calculus, we need to lift the operations of the theory to also act as constructors for predicates. We can then build up more significant formulae:

$$\texttt{prime} := \neg[\check{\texttt{e}}] \wedge \neg[\bar{\texttt{m}}(\neg[\check{\texttt{e}}], \neg[\check{\texttt{e}}])].$$

This lifting is just like extending an operation to act on subsets: take the product, and then apply the operation pointwise:

$$\begin{array}{ccc} \mathrm{Sub}(\mathtt{T}(-,\mathtt{M}))^2 & \overset{\bar{\mathtt{m}}}{\dashrightarrow} & \mathrm{Sub}(\mathtt{T}(-,\mathtt{M})) \\ {\scriptstyle\times}\downarrow & & \uparrow{\scriptstyle \mathrm{Sub}(\mathtt{T}(-,\mathtt{m}))} \\ \mathrm{Sub}(\mathtt{T}(-,\mathtt{M})^2) & \xrightarrow[\langle -,- \rangle]{\sim} & \mathrm{Sub}(\mathtt{T}(-,\mathtt{M}^2)). \end{array}$$

The Predicate Calculus

Theorem

Let $\mathtt{T}$ *be an algebraic theory with presentation* $(\mathcal{T}, \mathcal{O}, \mathcal{E})$*. Define the map*

$$\begin{aligned} \mathrm{pred}(\mathtt{T}) : \quad \mathtt{T} &\to \mathrm{HeyAlg} \\ \textstyle\prod_i \mathtt{t}_i &\mapsto \textstyle\prod_i \mathrm{Sub}(\hat{\mathtt{t}_i}) \\ [\mathtt{op} : \textstyle\prod_i^m \mathtt{t}_i \to \mathtt{t}] &\mapsto [\overline{\mathtt{op}} : \textstyle\prod_i^m \mathrm{Sub}(\hat{\mathtt{t}_i}) \to \mathrm{Sub}(\hat{\mathtt{t}})], \end{aligned}$$

where we define

$$\overline{\mathtt{op}}(\varphi_1, \ldots, \varphi_m)(\mathtt{s}) = \{\mathtt{op}(\mathtt{o}_1, \ldots, \mathtt{o}_m) \mid \forall i.\ \mathtt{o}_i \in \varphi_i(\mathtt{s})\}.$$

Then $\mathrm{pred}(\mathtt{T})$ *is a model of* $\mathtt{T}$*, in the category of Heyting algebras.*

Theories with Binding

To apply this idea to programming languages, we need a richer notion of theory which allows for operations to *bind variables*. This can be given by a systematic approach to contexts.

Λ

$$\frac{1 \leq i \leq n}{n \vdash x_i}\,[\nu] \qquad \frac{n \vdash t_0 \qquad n \vdash t_1}{n \vdash \mathsf{app}(t_0, t_1)}\,[@] \qquad \frac{n+1 \vdash t}{n \vdash \mathsf{abs}(x_{n+1}.t)}\,[\lambda]$$

This latter rule inputs a term with a distinguished free variable, and outputs a term with that variable bound. This is an operation whose arity is not a finite set, but an exponent with more structure:

$$\lambda : X^{\mathbb{V}} \Rightarrow X$$

$$\lambda_n : X^{\mathbb{V}}(n) = X(n+1) \to X(n).$$

Theories with Binding

Definition

A **second-order algebraic theory** $\langle \mathtt{T}; (\mathcal{T}, \mathcal{O}, \mathcal{E}); \mathtt{X} \rangle$ is an algebraic theory equipped with a set $\mathtt{X}$ of exponentiable objects. [2]

These theories generalize but remain faithful to universal algebra, with associated equational logic and correspondence to monads and operads.

We can lift binding operations using a similar method given in the theorem, and we have a model $\mathrm{pred}(\mathtt{T})$ which preserves exponentiable objects.

Theories with Binding

ρ calculus

$$
\begin{array}{llll}
\mathtt{0}: & 1 \to \mathtt{P} & \mathtt{in}: & \mathtt{N} \times [\mathtt{N} \Rightarrow \mathtt{P}] \to \mathtt{P} \\
\mathtt{@}: & \mathtt{P} \to \mathtt{N} & \mathtt{out}: & \mathtt{N} \times \mathtt{P} \to \mathtt{P} \\
*: & \mathtt{N} \to \mathtt{P} & \mathtt{par}: & \mathtt{P} \times \mathtt{P} \to \mathtt{P}
\end{array}
$$

$$
\begin{array}{ll}
[\mathtt{P}, \mathtt{par}, \mathtt{0}] & \mathtt{commutative\ monoid} \\
[\mathtt{reify}] & *(@(\mathtt{p})) = \mathtt{p} \\
[\mathtt{comm}] & \mathtt{par}(\mathtt{out}(\mathtt{a}, \mathtt{q}), \mathtt{in}(\mathtt{a}, \mathtt{p}[\mathtt{x}])) = \mathtt{p}\{@\mathtt{q}/\mathtt{x}\}
\end{array}
$$

Application Namespace Logic

Much of the complexity of distributed computing is in determining for many parallel processes a sequential evaluation.

In principle, many algorithms will want to analyze systems of "prime" processes, e.g. iteratively testing for race conditions.

prime $\simeq$ single-threaded

$$\texttt{single.thread} := \neg[\check{0}] \wedge \neg[\overline{\texttt{par}}(\neg[\check{0}], \neg[\check{0}])].$$

One could then build a predicate for "pairs of single-threaded processes which can communicate".

Application Namespace Logic

Let $\mathtt{T} := \mathrm{Th}.\rho$, and take $\mathrm{pred}(\mathtt{T}) : \mathtt{T} \to \mathrm{HeyAlg}$. This is really a **polymorphic type system**, with $\mathcal{T}$ as kinds, and $\mathit{im}(\mathrm{pred}(\mathtt{T}))$ as types.

Given $\mathtt{n} : 1 \to \mathtt{N}$ and $\Phi \rightarrowtail \hat{\mathtt{N}}$, define

$$[\mathtt{n} : \Phi] := [\mathtt{n} \in \Phi(1)].$$

The compatibility of the operations of the theory and the lifted operations on predicates gives that the former are "polymorphic":

$$\frac{\Gamma, \mathtt{x} : \Psi \vdash \mathtt{n} : \Phi,\ \mathtt{p}[\mathtt{x}] : \Psi \Rightarrow \Theta}{\Gamma \vdash \mathtt{in}(\mathtt{n}, \mathtt{p}[\mathtt{x}]) : \overline{\mathtt{in}}(\Phi, \Psi \Rightarrow \Theta).}\ [\mathtt{in}]$$

Application Namespace Logic

In fact we get even more: because the model preserves exponentials, we can now bind variables of *types*. The theory is now "parameterized".

This provides a structural query system which lets us begin any program with "receive from this namespace any code *of the following form*".

Structural Query

$$\frac{\Gamma, \mathtt{q} : \mathtt{P}, \Phi : \hat{\mathbb{N}} \vdash \mathtt{@q} : \Phi}{\Gamma \vdash \mathtt{par}(\mathtt{out}(\mathtt{n}, \mathtt{q}), \mathtt{in}(\mathtt{n}, \mathtt{p}[\mathtt{x} : \Phi])) \Rightarrow \mathtt{p}\{\mathtt{@q}/\mathtt{x}\} : \mathtt{P}} \; [\mathtt{COMM}_{\Phi}]$$

References


L.G. Meredith and Matthias Radestock,
Namespace Logic: A logic for a reflective higher-order calculus,
Trustworthy Global Computing, Edinburgh 2005.


Marcelo Fiore and Ola Mahmoud,
Second-Order Algebraic Theories,
Mathematical Foundations of Computer Science, Heidelberg 2010.