Computational Types, Collection Types and Monads
PhD course@UNIMI 2012-09
(last update September 12, 2012)

Eugenio Moggi

moggi@unige.it

DIBRIS, Univ. of Genova

PART 1

Denotational Semantics of Programming Languages
. . . via translation into a Metalanguage

Computational Types and Collection Types:
Informal Concepts and Simple Examples

Denotational Semantics [Scott-Strachey 1969]

Denotational Semantics is directly inspired by Mathematical Logic.

- The semantics of a (programming) language PL is given by an interpretation (the *intended model*) of its syntax in suitable mathematical structures (*domains*)
  - a type (syntactic category) τ of PL is interpreted by a domain $[\![\tau]\!]$
  - a term (syntactic entity) e of PL of type τ is interpreted by an *element* in $[\![\tau]\!]$
- The interpretation is *compositional*, i.e. it is defined by induction on the syntax
- The interpretation can be described by a translation from the (programming) language PL to a (mathematical) metalanguage ML, e.g. the language of Set Theory or some Typed Lambda Calculus

$$PL \xrightarrow{\quad transl \quad} ML \xrightarrow{\quad interp \quad} \mathcal{C}$$

In contrast, the Operational Semantics [Landin 1964] of PL is given by a (virtual) machine for executing terms of PL.

Note. Recursive definitions are pervasive in general purpose programming languages, but introduce semantic complications. However, the semantics of computational effects can be given independently from recursive definitions.

Denotational Semantics of a Simple Imperative Language

Let IMP be an imperative language with a global store, where each location $l \in L$ stores a natural number $n \in \mathrm{nat}$, and with the following syntactic categories

- expressions $e \in Exp$ of type nat, that cannot modify the store
- commands $c \in Cmd$, that can modify the store

For the semantics of IMP it suffices to take sets as domains.

Interpretation of Syntactic Categories

If expressions are always defined and commands have no loops, then

τ	$[\![\tau]\!]$
Exp	$S \longrightarrow \mathrm{nat}$, where $S \stackrel{\Delta}{=} L \to \mathrm{nat}$ is the set of stores
Cmd	$S \longrightarrow S$

Denotational Semantics of a Simple Imperative Language

Let IMP be an imperative language with a global store, where each location $l \in L$ stores a natural number $n \in \mathrm{nat}$, and with the following syntactic categories

- expressions $e \in Exp$ of type nat, that cannot modify the store
- commands $c \in Cmd$, that can modify the store

For the semantics of IMP it suffices to take sets as domains.

Interpretation of Syntactic Categories

If expressions can be ill-defined, but commands have no loops, then

τ	$[\![\tau]\!]$
Exp	$S \longrightarrow (\mathrm{nat} + \{err\})$, where $S \stackrel{\Delta}{=} L \to \mathrm{nat}$ is the set of stores
Cmd	$S \longrightarrow (S + \{err\})$

Denotational Semantics of a Simple Imperative Language

Let IMP be an imperative language with a global store, where each location $l \in L$ stores a natural number $n \in \mathrm{nat}$, and with the following syntactic categories

- expressions $e \in Exp$ of type nat, that cannot modify the store
- commands $c \in Cmd$, that can modify the store

For the semantics of IMP it suffices to take sets as domains.

Interpretation of Syntactic Categories

If expressions can be ill-defined and commands can have loops, then

τ	$[\![\tau]\!]$
Exp	$S \longrightarrow (\mathrm{nat} + \{err\})$, where $S \stackrel{\Delta}{=} L \to \mathrm{nat}$ is the set of stores
Cmd	$S \longrightarrow (S + \{err, div\})$

Denotational Semantics of a Simple Functional Language

Let PCF be a functional language with the following syntactic categories

- expressions $e \in Exp[\tau]$ of type τ, where $\tau ::= \mathrm{nat} \mid \tau_1 \to \tau_2$

For the semantics of PCF sets suffice, if recursive definitions are not allow. However, in the presence of recursive definitions, we need to use *cpos* (or similar structures).

Addendum on ω-cpos

An ω**-cpo** is a poset $(X, \leq)$ s.t. every ω**-chain** $(x_n | n \in \omega)$, i.e. $\forall n.x_n \leq x_{n+1}$, has a **least upper bound** (lub) denoted $\sqcup_n x_n$, i.e. $(\sqcup_n x_n) \leq y \iff (\forall n.x_n \leq y)$

- a set X *corresponds* to a **flat cpo** $(X, =)$
- lifting $X_\perp$ is the cpo obtained by adding a least element $\perp$ to the cpo X
- cartesian product $X \times Y$ and disjoint union $X + Y$ extend to cpos
- $X \longrightarrow Y$ is the cpo of ω**-continuous maps** f from cpo X to cpo Y, i.e.
  - f is monotonic, $x \leq y \implies f(x) \leq f(y)$, and
  - f preserves lubs of ω-chains, $f(\sqcup_n x_n) = \sqcup_n f(x_n)$
- if X is a cpo with a least element $\perp$, then every ω-continuous map $f : X \longrightarrow X$ has a **least fix-point** $x = f(x)$ given by $x \stackrel{\Delta}{=} \sqcup_n f^n(\perp)$.

Denotational Semantics of a Simple Functional Language

Let PCF be a functional language with the following syntactic categories

- expressions $e \in Exp[\tau]$ of type τ, where $\tau ::= \mathrm{nat} \mid \tau_1 \to \tau_2$

For the semantics of PCF sets suffice, if recursive definitions are not allow. However, in the presence of recursive definitions, we need to use *cpos* (or similar structures).

Interpretation of Syntactic Categories

If expressions do not include recursive definitions, then

τ	$[\![\tau]\!]$
$Exp[\mathrm{nat}]$	nat, i.e. the set of natural numbers
$Exp[\tau_1 \to \tau_2]$	$[\![Exp[\tau_1]]\!] \longrightarrow [\![Exp[\tau_2]]\!]$

Question. What should be the interpretation if expressions can be ill-defined?

Denotational Semantics of a Simple Functional Language

Let PCF be a functional language with the following syntactic categories

- expressions $e \in Exp[\tau]$ of type τ, where $\tau ::= \mathrm{nat} \mid \tau_1 \to \tau_2$

For the semantics of PCF sets suffice, if recursive definitions are not allow. However, in the presence of recursive definitions, we need to use *cpos* (or similar structures).

Interpretation of Syntactic Categories

If expressions include recursive definitions and parameters are not evaluated before a function call (CBN semantics), then

τ	$[\![\tau]\!]$
$Exp[\mathrm{nat}]$	$\mathrm{nat}_\perp$, where nat is the flat cpo of natural numbers
$Exp[\tau_1 \to \tau_2]$	$([\![Exp[\tau_1]]\!] \longrightarrow [\![Exp[\tau_2]]\!])_\perp$

Denotational Semantics of a Simple Functional Language

Let PCF be a functional language with the following syntactic categories

- expressions $e \in Exp[\tau]$ of type τ, where $\tau ::= \mathrm{nat} \mid \tau_1 \to \tau_2$

For the semantics of PCF sets suffice, if recursive definitions are not allow. However, in the presence of recursive definitions, we need to use *cpos* (or similar structures).

Interpretation of Syntactic Categories

If expressions include recursive definitions and parameters are evaluated before a function call (CBV semantics), then

τ	$[\![\tau]\!]$
$Exp[\mathrm{nat}]$	$\mathrm{nat}_\perp$, where nat is the flat cpo of natural numbers
$Exp[\tau_1 \to \tau_2]$	$([\![Exp[\tau_1]]\!] \xrightarrow[\perp]{} [\![Exp[\tau_2]]\!])_\perp$

where $(X_\perp \xrightarrow[\perp]{} Y_\perp)$ is the cpo of *strict* maps f in $X_\perp \longrightarrow Y_\perp$, i.e. $f(\perp) = \perp$.

$(- \xrightarrow[\perp]{} -)$ is a *clumsy* construction[a] , defined only on cpos with a least element.

However, $(X_\perp \xrightarrow[\perp]{} Y_\perp)$ is isomorphic to $X \longrightarrow Y_\perp$.

[a] The domains originally proposed by Scott had always a least element.

Addendum on Syntax and Semantics

Before giving the operational or denotation semantics of a PL, it can be convenient to restructure the syntax of PL, e.g. by introducing *auxiliary syntactic categories*.
Before giving the interpretation $[\![\tau]\!]$ of a type (syntactic category) of PL, it can be convenient to introduce *auxiliary semantic domains*.

Examples

- $S \triangleq L \to \mathrm{nat}$ is an auxiliary domain for giving the semantics of IMP
- Numeral expressions $Val \subset Exp$ can be introduced as an auxiliary syntactic category of IMP, s.t. $[\![Val]\!] = \mathrm{nat}$
- Value expressions $Val[\tau] \subset Exp[\tau]$ can be introduced as auxiliary syntactic categories of PCF, to give a *clearer* definition of the CBV and CBN semantics

τ	$[\![\tau]\!]$ in CBN	$[\![\tau]\!]$ in CBV
$Val[\mathrm{nat}]$	nat	like in CBN
$Exp[\tau]$	$[\![Val[\tau]]\!]_\perp$	like in CBN
$Val[\tau_1 \to \tau_2]$	$[\![Exp[\tau_1]]\!] \longrightarrow [\![Exp[\tau_2]]\!]$	$[\![Val[\tau_1]]\!] \longrightarrow [\![Exp[\tau_2]]\!]$

Computational Types [Mog91]

- Basic idea: distinguish the domain A for interpreting values (of type τ) from the domain MA for interpreting programs/expressions (of type τ). The unary constructor M is called a *notion of computation*.
- Categorical semantics: study M in the setting of a generic category $\mathcal{C}$, but concrete examples given in the category of sets or ω-cpos.
- Metalanguages: extends ML with computational types $M\tau$ and related terms
  - ret $\dfrac{e\colon \tau}{\mathrm{ret}\, e\colon M\tau}$ every value corresponds to a trivial computation
  - do $\dfrac{e_1\colon M\tau_1 \quad x\colon \tau_1 \vdash e_2\colon M\tau_2}{\mathrm{do}\,\{x \leftarrow e_1; e_2\}\colon M\tau_2}$ sequential execution of computations
  - fix $\dfrac{x\colon M\tau \vdash e\colon M\tau}{\mathrm{fix}\, x.e\colon M\tau}$ recursive definition of computations

$$PL \xrightarrow{\quad transl \quad} ML_M(\Sigma) \xrightarrow{\quad transl \quad} ML \longrightarrow \mathcal{C}$$

- Coherence: categorical semantics $\Longleftrightarrow$ syntax+equational theory of metalanguage

Examples of Computational Types/Notions of Computation

Simple examples in sets

- **partiality** $MA = A + \{\bot\}$, $\bot$ represents the *diverging computation*
- **nondeterminism** $MA = \mathcal{P}_{fin}(A)$ set of finite subsets of A , also a collection type
- **side-effects** $MA = (A \times S)^S$, S is a set of states, e.g. U^L or U^*
- **state-readers** $MA = A^S$, S is a set of states
- **exceptions** $MA = A + E$, E set of exceptions
- **continuations** $MA = R^{(R^A)}$, R set of results
- **interactive input** $MA = \mu X.A + X^U$, i.e. U-branching trees with A-labeled leaves, $\mu X.\tau$ is the *initial solution* to the domain equation $X = \tau$
- **interactive output** $MA = \mu X.A + (U \times X) \cong U^* \times A$
- **soft constrains** $MA = A \stackrel{fin}{\to} S$ where $S = (|S|, +, *, 0, 1)$ semi-ring, i.e. the set of functions $p\colon A \to |S|$ with finite support $\{a \in A | p(a) = 0\}$, also a collection type
- **algebraic terms** $MA = T_\Sigma(A)$ where Σ single-sorted algebraic signature, i.e. the set of Σ-terms with variables in A (generalizes to algebraic theory). See also *algebraic approach* to computational effects [PP01,HPP02,PP09].

Examples of Computational Types/Notions of Computation

Simple examples in cpos, MA has a least element, thus it supports recursive definitions of computations

- **partiality** $MA = A_\perp$, i.e. lifting
- **nondeterminism** $MA =$ *some powerdomain*
- **side-effects** $MA = (A \times S)_\perp{}^S$
- **state-readers** $MA = A_\perp{}^S$, S is a set of states
- **exceptions** $MA = (A + E)_\perp$
- **continuations** $MA = R^{(R^A)}$, R cpo with $\perp$
- **interactive input** $MA = \mu X.(A + X^U)_\perp$,
  $\mu X.\tau$ is the *initial solution* to the domain equation $X = \tau$
- **interactive output** $MA = \mu X.(A + (U \times X))_\perp$

Examples of Computational Types/Notions of Computation

More complex examples in sets

- $MA = ((A + E) \times S)^S$ imperative programs with exceptions
- $MA = ((A \times S) + E)^S$ imperative programs with exceptions,
  the state is lost when an exception is raised
- $MA = (A + E)^S$ expressions with errors/exceptions
- $MA = \mu X.\mathcal{P}_{fin}(A + (U \times X))$ nondeterministic interactive programs,
  related to synchronization trees up to strong bisimulation
- $MA = \mu X.\mathcal{P}_{fin}((A + X) \times S)^S$ parallel imperative programs,
  related to resumptions and small-step operational semantics

Collection Types [BNTW95], aka Bulk Types [WT91]

- Basic idea: $c \in MA$ is a *finite collection* of elements of A.
- Categorical semantics: a lot in common with the categorical semantics of computational types. Concrete examples given in the category of sets.
- DB intermediate languages (for source-to-source optimization) with collection types $M\tau$ and related terms
  - unit $\dfrac{e\colon \tau}{\{e\}\colon M\tau}$ singleton collection
  - flat $\dfrac{e\colon M(M\tau)}{\mathrm{flat}\, e\colon M\tau}$ flattening of a collection of collections
  - map $\dfrac{e_1\colon M\tau_1 \quad x\colon \tau_1 \vdash e_2\colon \tau_2}{\{e_2 | x \leftarrow e_1\}\colon M\tau_2}$ *comprehension* for collections
  - $0\colon M\tau$ empty collection and $+\colon M\tau \times M\tau \to M\tau$ union of two collections
  - $==\colon M\tau \times M\tau \to \mathrm{bool}$ test for equality of two collections

Note. equality of collections should be decidable, while equality of programs is not!

Examples of Collection Types

Standard examples in sets

- **sets** $MA = \mathcal{P}_{fin}(A)$ set of finite subsets of A
- **lists** $MA = A^*$ set of finite lists of A
- **bags** $MA = A \xrightarrow{fin} \mathrm{nat}$, i.e. the set of functions $p\colon A \to \mathrm{nat}$ with finite support $\{a \in A | p(a) = 0\}$
- **binary trees** $MA = \{0\} + (\mu X.A + (X \times X))$, i.e. the set consisting of binary trees with A-labeled leaves.

Counter-examples in Sets

- $MA = A + \{0\}$, the union of two collections is problematic.
- $MA = A^S$. When S is infinite, a collection can have infinitely many elements. When S is finite (but not empty), the empty collection and the union of two collections are problematic. When S is empty, there is only one collection.

PART 2

From Lambda Calculus to Category Theory [Scott 1980]
A Categorical Manifesto [Goguen 1991]

A Taste of Category Theory [Asperti-Longo 1991, Pierce 1991]
focused on Monads

Category Theory (CT) as a Pure Theory of functions [Sco80]

- Theories of functions (application/composition) in alternative to Set Theory:
  1920s Combinatory Logic by Schönfinkel (1987-1889) and Curry (1900-1982)
  1930s Lambda Calculus by Church (1903-1995)
  1940s Category Theory[a] by Eilenberg (1913-1998) and MacLane (1909-2005).
- CT as a copernican change of view w.r.t. Set Theory:
  - how sets/classes are build from other sets/classes vs
  - how objects relate to other objects (in the same category).

  Derogative nickname of CT *abstract nonsense*.
- [Sco80] (Curry's 80th birthday): CT as a *milk-and-water* theory of functions
  - Dana Scott (1932-) PhD student of Church, ACM Turing Award 1976
  - Emeritus Professor of Comp. Sci., Phil. and Math. Logic at CMU

  axiomatic freedom (offered by CT) + naive view (using the *right spectacles*), e.g. every model of the extensional untyped lambda calculus is a *reflexive object* $(U \to U) \cong U$ in a CCC $\mathcal{C}$ ($\neq$ the category of sets).

Issue. What is the minimal categorical setting to study computational/collection types?

[a] Initial motivations from Algebraic Topology, rather than foundationals.

A Categorical Manifesto [Gog91]

Why Category Theory can be useful in Computer Science (or in a young subject, poorly organized, that needs all the help that it can get):

- Formulating definitions and theories (CT provides guidelines)
- Discovering and exploiting relations with other fields

  sufficiently abstract formulations can reveal surprising connections
- Dealing with abstraction and representation independence
- Formulating conjectures and research directions,
  mainly through relations with other fields
- Conceptual unification (by abstraction and use of few fundamental concepts)

CT useful also in a mature subject (e.g. to export ideas to other subjects):

- more general/abstract reformulations or cleaner/unified reformulations.

[Asperti-Longo 1991, Ch 1]

Category, Graph and Diagram

A *category* $\mathcal{C}$ consists of

- a collection $\mathcal{C}_0$ of objects, notation $\boxed{a \in \mathcal{C}}$
- a collection $\mathcal{C}_1$ of morphisms (arrows, maps)
- operations $\mathrm{dom}, \mathrm{cod}\colon \mathcal{C}_1 \longrightarrow \mathcal{C}_0$ assigning to each arrow a domain and codomain

  we write $\boxed{f \in \mathcal{C}[a,b] \text{ or } a \xrightarrow{f} b \text{ or } f\colon a \longrightarrow b \text{ when } a = \mathrm{dom}(f) \text{ and } b = \mathrm{cod}(f)}$
- an operation $\mathrm{id}\colon \mathcal{C}_0 \longrightarrow \mathcal{C}_1$ assigning to each object a an identity $\boxed{\mathrm{id}_a \in \mathcal{C}[a,a]}$
- a *composition* operation $\circ$ assigning to each pair f and g of *composable* arrows $\boxed{a \xrightarrow{f} b \xrightarrow{g} c}$ a composite arrow $\boxed{g \circ f \in \mathcal{C}[a,c]}$

and identity and composition satisfy the following properties

(identity) $\boxed{\mathrm{id}_b \circ f = f = f \circ \mathrm{id}_a}$ for any $\boxed{a \xrightarrow{f} b}$

(associativity) $\boxed{h \circ (g \circ f) = (h \circ g) \circ f}$ for any $\boxed{a \xrightarrow{f} b \xrightarrow{g} c \xrightarrow{h} d}$

Category, Graph and Diagram

- A *graph*[a] $\mathcal{G}$ consists of
  - a collection $\mathcal{G}_0$ of nodes (vertexes)
  - a collection $\mathcal{G}_1$ of arcs (edges, arrows)
  - operations $\mathrm{dom}, \mathrm{cod}: \mathcal{G}_1 \longrightarrow \mathcal{G}_0$ assigning to each arc a source and target

  we write $\boxed{a \xrightarrow{f} b \text{ when } a = \mathrm{dom}(f) \text{ and } b = \mathrm{cod}(f)}$

  Any category $\mathcal{C}$ has an underlying graph $\mathrm{dom}, \mathrm{cod}: \mathcal{C}_1 \longrightarrow \mathcal{C}_0$
- **Graph** is the category of *small* graphs (i.e. $\mathcal{G}_0$ and $\mathcal{G}_1$ are sets) with arrows

  $(g_0, g_1) \in$ **Graph**$[\mathcal{G}, \mathcal{G}'] \overset{\Delta}{\Longleftrightarrow} g_0: \mathcal{G}_0 \longrightarrow \mathcal{G}'_0$ and $g_1: \mathcal{G}_1 \longrightarrow \mathcal{G}'_1$ s.t.

  $\boxed{a \xrightarrow{f} b \text{ in } \mathcal{G} \text{ implies } g_0(a) \xrightarrow{g_1(f)} g_0(b) \text{ in } \mathcal{G}'}$

---

[a] In Graph Theory what we call graph is called a *directed multi-graph*.

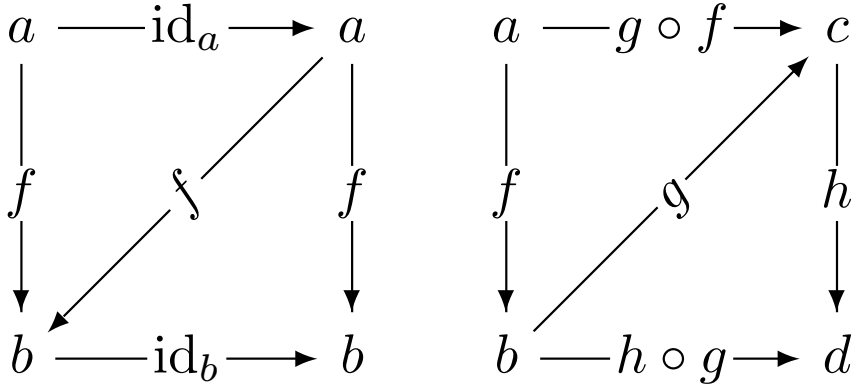
Category, Graph and Diagram

- Given a category $\mathcal{C}$ and a small graph $\mathcal{G}$ a *diagram* D of shape $\mathcal{G}$ in $\mathcal{C}$ is a graph morphism (d_0, d_1) from $\mathcal{G}$ to the underlying graph of $\mathcal{C}$, i.e. D corresponds to a consistent labeling of nodes and arcs of $\mathcal{G}$ with objects and arrows of $\mathcal{C}$

  given a *path* $p = (a_i \xrightarrow{f_i} a_{i+1} | i < n)$ from a_0 to a_n in $\mathcal{G}$ we write $D[p]$ for the arrow in $\mathcal{C}[d_0(a_0), d_0(a_n)]$ obtained by composing the arrows $d_1(f_i)$ (when $n = 0$ then $D[p]$ is the identity on $d_0(a_0)$)

- A diagram D *commutes* $\overset{\Delta}{\Longleftrightarrow}$ for every pair of paths p and p' in $\mathcal{G}$ with the same source and target (say a to b) $D[p] = D[p']$ (as arrows in $\mathcal{C}[d_0(a), d_0(b)]$)

  commuting diagrams expressing the (identity) and (associativity) properties



Examples

Dogma 1 [Gog91]: to each species of mathematical structure, there corresponds a **category** whose objects have that structure, and whose morphisms preserve it.

$\mathcal{C}$	Objects a	Morphisms $f \in \mathcal{C}[a_1, a_2]$
Set	sets X	functions $f \in X_1 \longrightarrow X_2$
	to be precise morphisms are triples (X_1, f, X_2)	
Inc	sets X	inclusions (functions)
Rel	sets X	relations $R \subseteq X_1 \times X_2$
Set, **Inc** and **Rel** same objects, but different morphisms		
Mon	monoids $(X, \cdot, 1)$ $x \cdot 1 = x = 1 \cdot x \quad (x_1 \cdot x_2) \cdot x_3 = x_1 \cdot (x_2 \cdot x_3)$	homomorphisms $f\colon X_1 \longrightarrow X_2$ $f(1_1) = 1_2 \quad f(x_1 \cdot_1 x_2) = f(x_1) \cdot_2 f(x_2)$
PO	partial orders $(X, \leq)$	monotone maps $f\colon X_1 \longrightarrow X_2$ $x_1 \leq_1 x_2 \implies f(x_1) \leq_2 f(x_2)$
ω-**CPO**	ω-cpos $(X, \leq)$	ω-continuous maps $f\colon X_1 \longrightarrow X_2$

Examples

- a collection C induces a *discrete* category $\mathcal{C}$ (i.e. every arrow is an identity):
  $\mathcal{C}_0 = \mathcal{C}_1 = C$ and $\mathrm{dom}(a) = a = \mathrm{cod}(a)$
- a *preorder* $(X, \leq)$, i.e. $\leq \subseteq X \times X$ is reflexive and transitive, induces a category $\mathcal{C}$ where every $\mathcal{C}[a, b]$ has at most one element:
  $\mathcal{C}_0 = X$, $\mathcal{C}_1 = \leq$, $\mathrm{dom}(a, b) = a$ and $\mathrm{cod}(a, b) = b$
  - $\subseteq$ is a preorder on sets (indeed a partial order)
  - $\in$ is not a preorder on sets (e.g. $X \in X$ fails in ZF set theory)
- a *monoid* $(X, \cdot, 1)$, induces a category $\mathcal{C}$ with exactly one object:
  $\mathcal{C}_0 = \{*\}$, $\mathcal{C}_1 = X$, $\mathrm{id}_* = 1$ and $x_1 \circ x_2 = x_1 \cdot x_2$

Examples from Algebra

Let Σ be an *algebraic signature*, i.e. a family $(\Sigma_n|n)$ of sets (of operator symbols) indexed by natural numbers (considered as *arities*)

- $T_\Sigma(X)$ denotes the set of Σ-terms with variables included in the set X

  T_Σ is the category of (finite) sets and *substitutions* $T_\Sigma[X_1, X_2] \stackrel{\Delta}{=} X_2 \longrightarrow T_\Sigma(X_1)$ given $\rho_1 \colon X_2 \longrightarrow T_\Sigma(X_1)$ and $\rho_2 \colon X_3 \longrightarrow T_\Sigma(X_2)$, the composite $\rho_2 \circ \rho_1$ is the $\rho \colon X_3 \longrightarrow T_\Sigma(X_1)$ s.t. $\rho(x) \stackrel{\Delta}{=} t[\rho_1]$ with $t = \rho_2(x) \in T_\Sigma(X_2)$

- a Σ-algebra is a pair $(A, [\![-]\!])$, where A is a set and $[\![-]\!]$ is an interpretation of the operator symbols in A, i.e. $[\![op]\!] \colon A^n \longrightarrow A$ for $op \in \Sigma_n$

  $\mathbf{Alg}_\Sigma$ is the category of Σ-algebras and Σ-*homomorphisms*[a]

$$\begin{array}{ccc} A_1^n & \xrightarrow{f^n} & A_2^n \\ \downarrow{\scriptstyle [\![op]\!]_1} & & \downarrow{\scriptstyle [\![op]\!]_2} \\ A_1 & \xrightarrow{f} & A_2 \end{array}$$

One can define also the categories $T_{(\Sigma,E)}$ and $\mathbf{Alg}_{(\Sigma,E)}$, with E set of Σ-equations.

[a] See [AspertiLongo91, Sec 4.1]

Examples from Computability

- **EN** is the category of *numbered sets*

  (objects) $\underline{X} = (X, e)$ with $e\colon N \twoheadrightarrow X$ (i.e. e onto map)

  (arrows) $\underline{X}_1 \xrightarrow{f} \underline{X}_2 \overset{\Delta}{\iff}$ exists a recursive $f'\colon N \to N$ s.t.

$$\begin{array}{ccc} X_1 & \xrightarrow{f} & X_2 \\ \uparrow{\scriptstyle e_1} & & \uparrow{\scriptstyle e_2} \\ N & \xrightarrow{f'} & N \end{array}$$

- Let $\underline{A} = (A, \cdot)$ be a *partial Combinatory Algebra*, i.e. $\cdot$ is a partial application and
  - exist $K, S \in A$ s.t. $K\ a\ b = a$, $S\ a\ b \downarrow$ and $S\ a\ b\ c \simeq a\ c\ (b\ c)$ for any $a, b, c \in A$
- $\underline{A}$**-Set** is the category of sets with an $\underline{A}$*-realizability* relation

  (objects) $\underline{X} = (X, \Vdash)$ with $\Vdash\ \subseteq A \times X$ *onto* $\boxed{\forall x \in X.\exists a.a \Vdash x}$

  (arrows) $\underline{X}_1 \xrightarrow{f} \underline{X}_2 \overset{\Delta}{\iff} X_1 \xrightarrow{f} X_2$ has a *realizer* r $\boxed{a \Vdash_1 x \text{ implies } r\ a \Vdash_2 f(x)}$

Examples from Category Theory

- The category **Cat** whose objects are (small) categories (by Dogma 1)[a]
- the *dual*[b] category $\mathcal{C}^{op}$ of $\mathcal{C}$: $\mathcal{C}_0^{op} = \mathcal{C}_0$ and $\boxed{\mathcal{C}^{op}[a,b] = \mathcal{C}[b,a]}$

  $\mathrm{id}_a^{op} = \mathrm{id}_a$ and $g \circ^{op} f = f \circ g$
- the *product* category $\mathcal{C} \times \mathcal{D}$ of $\mathcal{C}$ and $\mathcal{D}$: $(\mathcal{C} \times \mathcal{D})_0 = \mathcal{C}_0 \times \mathcal{D}_0$ and $(\mathcal{C} \times \mathcal{D})_1 = \mathcal{C}_1 \times \mathcal{D}_1$

  $\mathrm{id}_{(a,a')} = (\mathrm{id}_a, \mathrm{id}_{a'})$ and $(g, g') \circ (f, f') = (g \circ f, g' \circ f')$
- A category $\mathcal{D}$ is a *subcategory* of $\mathcal{C} \overset{\Delta}{\Longleftrightarrow} \mathcal{D}_0 \subseteq \mathcal{C}_0$ and $\mathcal{D}[a,b] \subseteq \mathcal{C}[a,b]$, and identities and composition in $\mathcal{D}$ coincide with those in $\mathcal{C}$

  $\mathcal{D}$ is a *full* subcategory when in addition $\mathcal{D}[a,b] = \mathcal{C}[a,b]$

  Set is a subcategory of **Rel** (but it is not full), since functions are relations (with certain properties)

  $\mathbf{Alg}_{(\Sigma,E)}$ is a full subcategory of $\mathbf{Alg}_{\Sigma}$, and also of $\mathbf{Alg}_{(\Sigma,E')}$ when $E' \subset E$

  T_Σ is a *basically* a full subcategory of $\mathbf{Alg}_{\Sigma}$, as $T_\Sigma(X)$ is the carrier of a Σ-algebra.

---

[a] Morphisms in **Cat** are functors, see [AspertiLongo91, Def 3.1.1]

[b] Duality is a powerful technique of Theory applicable to definitions and theorems.

Special Morphisms

Given a category $\mathcal{C}$ we say that

- $a \xrightarrow{e} b$ is *epic* $\overset{\Delta}{\iff}$ $f \circ e = g \circ e$ implies $f = g$ when $c \in \mathcal{C}$ and $f, g \in \mathcal{C}[b, c]$
- $a \xrightarrow{m} b$ is *monic* $\overset{\Delta}{\iff}$ $m \circ f = m \circ g$ implies $f = g$ when $c \in \mathcal{C}$ and $f, g \in \mathcal{C}[c, a]$

  monic and epic are dual properties, i.e. m is monic in $\mathcal{C} \iff m$ is epic in $\mathcal{C}^{op}$
- $a \xrightarrow{i} b$ is *iso* $\overset{\Delta}{\iff}$ $j \circ i = \mathrm{id}_a$ and $i \circ j = \mathrm{id}_b$ for some (unique) $j \in \mathcal{C}[b, a]$

  iso is a self-dual property, i.e. i is iso in $\mathcal{C} \iff i$ is iso in $\mathcal{C}^{op}$
- $a \xrightarrow{e} b$ is a *split epic* $\overset{\Delta}{\iff}$ $e \circ m = \mathrm{id}_b$ for some $m \in \mathcal{C}[b, a]$

  there is a dual property of *split monic*

The following statements and their dual hold (proofs are by *diagram chasing*):

- e split epic $\implies$ e epic
- m monic and split epic $\implies$ m iso

we write $a \xrightarrow{m} b$ when m is monic and $a \xrightarrow{e} b$ when e is epic

Special Morphisms

In **Set** one has the following concrete characterizations

- e epic $\Longleftrightarrow$ e is surjective $\Longleftrightarrow$ e split epic (by the axiom of choice)
- m monic $\Longleftrightarrow$ m is injective ($m\colon a \longrightarrow b$ split monic $\Longleftrightarrow$ m monic and $a \neq \emptyset$)
- i iso $\Longleftrightarrow$ i is bijective

Give concrete characterizations in other sample categories, in particular consider

- $\mathcal{C}$ is a monoid, i.e. a category with exactly one object
- $\mathcal{C}$ is a preorder (every arrow is both monic and epic)

[Asperti-Longo 1991, Ch 2]

Thinking Categorically (special objects)

- $0 \in \mathcal{C}$ *initial* $\overset{\Delta}{\Longleftrightarrow}$ $\boxed{\forall a \in \mathcal{C}.\exists! f \in \mathcal{C}[0, a]}$ $\quad$ $1 \in \mathcal{C}$ *terminal* $\overset{\Delta}{\Longleftrightarrow}$ $\boxed{\forall a \in \mathcal{C}.\exists! f \in \mathcal{C}[a, 1]}$

  initial and terminal are dual properties, i.e. a is terminal in $\mathcal{C} \Longleftrightarrow a$ is initial in $\mathcal{C}^{op}$

The following statements say that initial objects are determined up to unique iso

- if 0 is initial and $0 \xrightarrow{i} 0'$ is an iso, then $0'$ is initial
- if 0 and $0'$ are initial, then they are isomorphic $\boxed{\text{and the iso is unique}}$

In **Set** one has the following concrete characterizations

- X is initial $\Longleftrightarrow X = \emptyset$ ($\emptyset$ is both initial and terminal in **Rel** and **pSet)**
- X is terminal $\Longleftrightarrow X$ has exactly one element

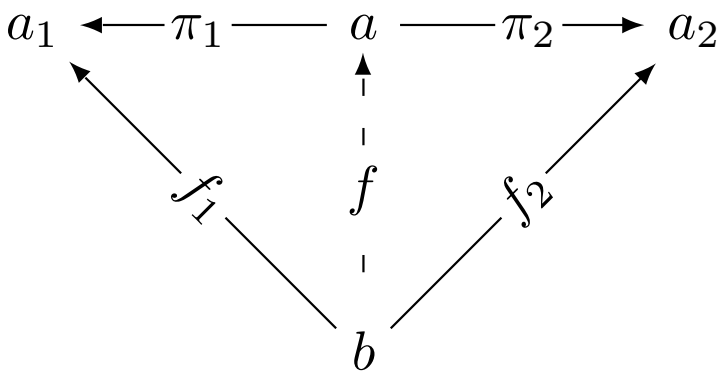
The property of being initial/terminal is a simple form of *universal property*, i.e.

- a *property* $P(x)$ expressed in the language of Category Theory, s.t.
- the *structures* x satisfying the property are determined up to unique iso

$\boxed{\text{thus the structures on which } P(x) \text{ is defined are the objects of a category}}$

Thinking Categorically (universal properties I)

- $a_1 \xleftarrow{\pi_1} a \xrightarrow{\pi_2} a_2$ is a *product* diagram in $\mathcal{C}$ $\overset{\Delta}{\Longleftrightarrow}$

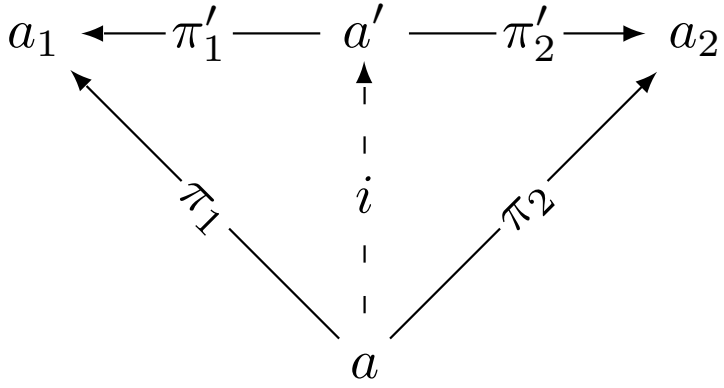


for any $a_1 \xleftarrow{f_1} b \xrightarrow{f_2} a_2$ $\exists! f$ s.t. commutes

we write $a_1 \times a_2$ for a and (f_1, f_2) for f

- product diagrams for a_1 and a_2 are determined up to unique iso, i.e.

if $a_1 \xleftarrow{\pi'_1} a' \xrightarrow{\pi'_2} a_2$ is another product diagram, then



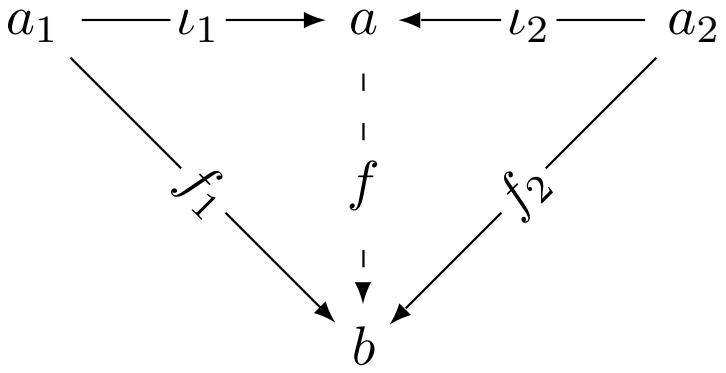
$\exists! i$ iso s.t. commutes

product diagrams are objects in the category of *cones* $a_1 \xleftarrow{f_1} b \xrightarrow{f_2} a_2$

Thinking Categorically (universal properties I)

- a *coproduct* diagram $a_1 \xrightarrow{\iota_1} a \xleftarrow{\iota_2} a_2$ is the dual of a product diagram, i.e.

for any $a_1 \xrightarrow{f_1} b \xleftarrow{f_2} a_2$ $\exists! f$ s.t.



commutes

we write $\boxed{a_1 + a_2 \text{ for } a \text{ and } [f_1, f_2] \text{ for } f}$

- coproduct diagrams for a_1 and a_2 are determined up to unique iso (by duality)

The notion of product/coproduct diagram generalizes to the I-*indexed* case (with I set)

- when $I = \emptyset$ the definitions coincide with that of terminal and initial object
- the notation introduced for binary products and coproducts is modified as follows

$\boxed{\prod_{i \in I} a_i \text{ and } \coprod_{i \in I} a_i}$ and $\boxed{(f_i | i \in I) \text{ and } [f_i | i \in I]}$

Thinking Categorically (universal properties I)

In **Set** for any pair of object X_1 and X_2 we have that

- $X_1 \xleftarrow{\pi_1} X_1 \times X_2 \xrightarrow{\pi_2} X_2$ is a product diagram, where $X_1 \times X_2$ is the cartesian product and $\pi_i(x_1, x_2) = x_i$
- $X_1 \xrightarrow{\iota_1} X_1 \uplus X_2 \xleftarrow{\iota_2} X_2$ is a coproduct diagram, where $X_1 \uplus X_2$ is the disjoint union $\{(i, x) | x \in X_i\}$ and $\iota_i(x) = (i, x)$

In **Rel** $X_1 \uplus X_2$ is both the product and the coproduct of X_1 and X_2. In **Inc** the product is $X_1 \cap X_2$ and the coproduct is $X_1 \cup X_2$. When $\mathcal{C}$ is a preorder one has

- an initial object 0 is a least element $\bot$, and a terminal object 1 is a top element $\top$
- a product $a_1 \times a_2$ is a greatest lower bound $a_1 \wedge a_2$, and a coproduct $a_1 + a_2$ is a least upper bound $a_1 \vee a_2$

When the objects involved exist, there are *canonical* isomorphisms

- $a \times 1 \cong a \quad a_1 \times a_2 \cong a_2 \times a_1 \quad (a_1 \times a_2) \times a_3 \cong a_1 \times (a_2 \times a_3)$
- similar isomorphisms hold by replacing $\times$ with $+$ and 1 with 0

In **Set** (in biCCCs, but not in general) the *canonical* maps below are iso

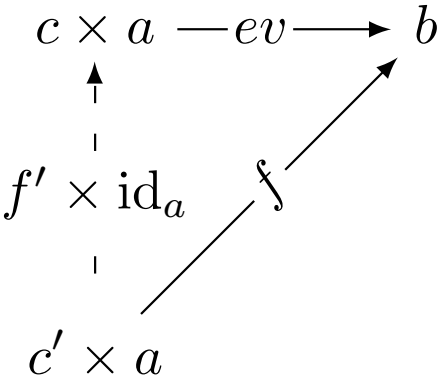
- $0 \dashrightarrow a \times 0 \qquad (a \times a_1) + (a \times a_2) \dashrightarrow a \times (a_1 + a_2)$

Thinking Categorically (universal properties II)

Given a category with binary products

- $c \times a \xrightarrow{ev} b$ is an *exponential* diagram in $\mathcal{C} \overset{\Delta}{\Longleftrightarrow}$

  for any $c' \times a \xrightarrow{f} b$ $\exists! f' : c' \to c$ s.t. $f' \times \mathrm{id}_a$ commutes



  where $f' \times \mathrm{id}_a \overset{\Delta}{=} (f' \circ \pi_1, \pi_2)$, we write b^a for c and $\Lambda(f)$ for f'

- exponential diagrams are determined up to unique iso
- In **Set** an exponential diagram is $Y^X \times X \xrightarrow{ev} Y$, where Y^X is the set of functions **Set**$[X, Y]$ and $ev(f, x) = f(x)$
- In ω-**CPO** an exponential diagram is $\underline{Y}^{\underline{X}} \times \underline{X} \xrightarrow{ev} \underline{Y}$, where $\underline{Y}^{\underline{X}}$ is the set of ω-continuous maps ω-**CPO**$[\underline{X}, \underline{Y}]$ with the pointwise order
- In $\underline{A}$-**Set** an exponential diagram is $\underline{Y}^{\underline{X}} \times \underline{X} \xrightarrow{ev} \underline{Y}$, where $\underline{Y}^{\underline{X}}$ is the set of realizable maps $\underline{A}$-**Set**$[\underline{X}, \underline{Y}]$ with an *obvious* realizability relation

Thinking Categorically (universal properties II)

- $\mathcal{C}$ has *enough points* $\overset{\Delta}{\Longleftrightarrow}$ it has a terminal object 1 and for any $f, g \in \mathcal{C}[a, b]$ $(\forall x\colon 1 \to a. f \circ x = g \circ x) \implies f = g$
- $\mathcal{C}$ is a *cartesian* category (CC for short) $\overset{\Delta}{\Longleftrightarrow}$ it has a terminal object 1 and binary products $a_1 \times a_2$ for any pair of objects
- $\mathcal{C}$ is a *cartesian closed* category (CCC for short) $\overset{\Delta}{\Longleftrightarrow}$ it is cartesian and it has exponentials b^a for any pair of objects
- $\mathcal{C}$ is a *bi-cartesian closed* category (biCCC for short) $\overset{\Delta}{\Longleftrightarrow}$ it is cartesian closed and it has *finite coproducts* In a biCCC $\coprod_{i \in n} (a \times a_i) \dashrightarrow a \times (\coprod_{i \in n} a_i)$ is an iso.

Set, **PO**, ω**-CPO**, $\underline{A}$**-Set**, **Cat** are biCCC. **Inc**, **Rel**, **Graph**, **Mon**, **EN** are not CCC.

Equational reformulation

- $\pi_i \circ (f_1, f_2) = f_i$ and $(\pi_1 \circ f, \pi_2 \circ f) = f\colon b \longrightarrow a_1 \times a_2$
- $[f_1, f_2] \circ \iota_i = f_i$ and $[f \circ \iota_1, f \circ \iota_2] = f\colon a_1 + a_2 \longrightarrow b$
- $ev \circ (a \times \Lambda(f)) = f$ and $\Lambda(ev \circ (f' \times \mathrm{id}_a)) = f'\colon c' \longrightarrow b^a$

Addendum: Internal Language ML for Categories

The *internal language* ML of a category $\mathcal{C}$ consists of

- types $\tau ::= a \mid \ldots$ and contexts $\Gamma ::= x{:}\tau \mid \ldots$, with a object of $\mathcal{C}$ and x variable
- raw terms $e ::= x \mid f(e) \mid \ldots$ with f arrow of $\mathcal{C}$, and several judgments
  - $\Gamma \vdash e{:}\tau$ asserting well-formedness of term e

$$x \ \frac{}{x{:}\tau \vdash x{:}\tau} \qquad f \ \frac{\Gamma \vdash e{:}\tau}{\Gamma \vdash f(e){:}\tau'} \ [\![\tau]\!] \xrightarrow{f} [\![\tau']\!]$$

  - $\Gamma \vdash e_1 = e_2{:}\tau$ asserting equality of well-formed terms

The interpretation $[\![-]\!]$ of ML in $\mathcal{C}$ goes a follows

- types τ and contexts Γ are interpreted by objects of $\mathcal{C}$ $\boxed{[\![a]\!] \stackrel{\Delta}{=} a \quad [\![x{:}\tau]\!] \stackrel{\Delta}{=} [\![\tau]\!]}$
- well-formed terms $\Gamma \vdash e{:}\tau$ are interpreted[a] by arrows $f{:}[\![\Gamma]\!] \longrightarrow [\![\tau]\!]$

  $\boxed{[\![x{:}\tau \vdash x{:}\tau]\!] \stackrel{\Delta}{=} \mathrm{id}_a}$ with $a = [\![\tau]\!]$ and $\boxed{[\![\Gamma \vdash f(e){:}\tau']\!] \stackrel{\Delta}{=} f \circ [\![\Gamma \vdash e{:}\tau]\!]}$
- equality judgments are interpreted by equality of arrows.

[a] the interpretation is defined by induction on the *unique* derivation of $\Gamma \vdash e{:}\tau$.

Addendum: Internal Language ML for Categories

The *internal language* ML of a category $\mathcal{C}$ consists of

- types $\tau ::= a \mid \ldots$ and contexts $\Gamma ::= x{:}\tau \mid \ldots$, with a object of $\mathcal{C}$ and x variable
- raw terms $e ::= x \mid f(e) \mid \ldots$ with f arrow of $\mathcal{C}$, and several judgments
  - $\Gamma \vdash e{:}\tau$ asserting well-formedness of term e

$$x \ \frac{}{x{:}\tau \vdash x{:}\tau} \qquad f \ \frac{\Gamma \vdash e{:}\tau}{\Gamma \vdash f(e){:}\tau'} \ [\![\tau]\!] \xrightarrow{f} [\![\tau']\!]$$

  - $\Gamma \vdash e_1 = e_2{:}\tau$ asserting equality of well-formed terms

Substitution is Composition

- subst $\dfrac{\Gamma \vdash e{:}\tau \qquad x{:}\tau \vdash e'{:}\tau'}{\Gamma \vdash e'[x{:}e]{:}\tau'}$ is an admissible rule
- $[\![\Gamma \vdash e'[x{:}e]{:}\tau']\!] = g \circ f$ if $[\![\Gamma \vdash e{:}\tau]\!] = c \xrightarrow{f} a$ and $[\![x{:}\tau \vdash e'{:}\tau']\!] = a \xrightarrow{g} b$

Addendum: Internal Language ML for Categories

The *internal language* ML of a category $\mathcal{C}$ consists of

- types $\tau ::= a \mid \ldots$ and contexts $\Gamma ::= x{:}\tau \mid \ldots$, with a object of $\mathcal{C}$ and x variable
- raw terms $e ::= x \mid f(e) \mid \ldots$ with f arrow of $\mathcal{C}$, and several judgments
  - $\Gamma \vdash e{:}\tau$ asserting well-formedness of term e

$$x \ \frac{}{x{:}\tau \vdash x{:}\tau} \qquad f \ \frac{\Gamma \vdash e{:}\tau}{\Gamma \vdash f(e){:}\tau'} \ [\![\tau]\!] \xrightarrow{f} [\![\tau']\!]$$

  - $\Gamma \vdash e_1 = e_2{:}\tau$ asserting equality of well-formed terms

Equality of Terms

$$\frac{\Gamma \vdash e{:}\tau}{\Gamma \vdash e = e{:}\tau} \qquad \frac{\Gamma \vdash e_1 = e_2{:}\tau}{\Gamma \vdash e_2 = e_1{:}\tau} \qquad \frac{\Gamma \vdash e_1 = e_2{:}\tau \qquad \Gamma \vdash e_2 = e_3{:}\tau}{\Gamma \vdash e_1 = e_3{:}\tau}$$

$$\text{congr} \ \frac{\Gamma \vdash e_1 = e_2{:}\tau \qquad x{:}\tau \vdash e{:}\tau'}{\Gamma \vdash e[x{:}e_1] = e[x{:}e_2]{:}\tau'} \qquad \text{subst} \ \frac{\Gamma \vdash e{:}\tau \qquad x{:}\tau \vdash e_1 = e_2{:}\tau'}{\Gamma \vdash e_1[x{:}e] = e_2[x{:}e]{:}\tau'}$$

$$\text{id} \ \frac{}{x{:}\tau \vdash x = \mathrm{id}_a(x){:}\tau} \ a = [\![\tau]\!] \qquad \text{comp} \ \frac{}{x{:}\tau \vdash h(x) = g(f(x)){:}\tau'} \ h = [\![\tau]\!] \xrightarrow{f} \xrightarrow{g} [\![\tau']\!]$$

Addendum: Internal Language $ML_\times$ for Cartesian Categories

- types $\tau ::= a \mid 1 \mid \tau_1 \times \tau_2$ and contexts $\Gamma ::= 1 \mid \Gamma, x{:}\tau$ with $x \notin \Gamma$
- raw terms $e ::= x \mid f(e) \mid () \mid (e_1, e_2) \mid \pi_1(e) \mid \pi_2(e)$
- additional rules for well-formedness of terms

$$\frac{x{:}\tau \in \Gamma}{\Gamma \vdash x{:}\tau} \qquad \frac{}{\Gamma \vdash ():1} \qquad \frac{\Gamma \vdash e_1{:}\tau_1 \qquad \Gamma \vdash e_2{:}\tau_2}{\Gamma \vdash (e_1, e_2){:}\tau_1 \times \tau_2} \qquad \frac{\Gamma \vdash e{:}\tau_1 \times \tau_2}{\Gamma \vdash \pi_i(e){:}\tau_i}$$

Interpretation of types and terms require a choice of product diagrams.

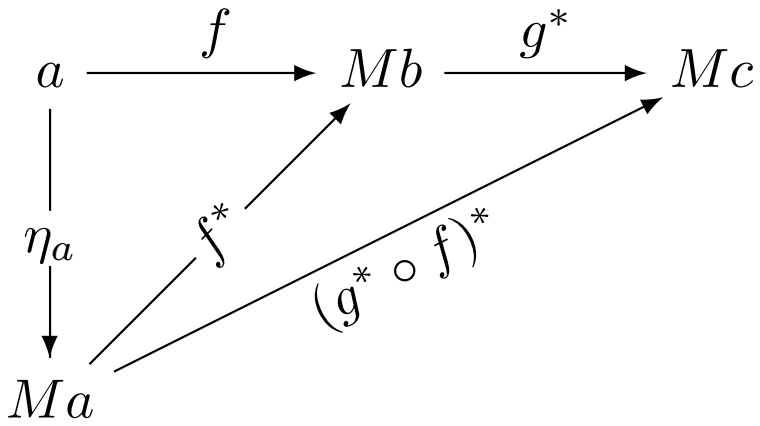
- substitution subst_{par} $\dfrac{\{\Gamma \vdash e_i{:}\tau_i \mid i \in n\} \quad (x_i{:}\tau_i | i \in n) \vdash e'{:}\tau'}{\Gamma \vdash e'[x_i{:}e_i | i \in n]{:}\tau'}$ subst_{inc} $\dfrac{\Gamma \vdash e{:}\tau \quad \Gamma, x{:}\tau \vdash e'{:}\tau'}{\Gamma \vdash e'[x{:}e]{:}\tau'}$
- additional rules for equality of terms

$$\frac{\Gamma \vdash e{:}1}{\Gamma \vdash e = (){:}1} \qquad \frac{\Gamma \vdash e_1{:}\tau_1 \qquad \Gamma \vdash e_2{:}\tau_2}{\Gamma \vdash \pi_i(e_1, e_2) = e_i{:}\tau_i} \qquad \frac{\Gamma \vdash e{:}\tau_1 \times \tau_2}{\Gamma \vdash e = (\pi_1(e), \pi_2(e)){:}\tau_1 \times \tau_2}$$

Monad M on $\mathcal{C}$ in extension form $(M, \eta, \_^*)$ [Man76]

- $M : \mathcal{C}_0 \longrightarrow \mathcal{C}_0$ $\qquad \dfrac{a \in \mathcal{C}_0}{a \xrightarrow{\eta_a} Ma} \qquad \dfrac{a \xrightarrow{f} Mb}{Ma \xrightarrow{f^*} Mb}$ s.t.

- $Ma \underset{\mathrm{id}_{Ma}}{\overset{\eta_a^*}{\rightrightarrows}} Ma$



when $b \xrightarrow{g} Mc$

The Kleisli category $\mathcal{C}_M$ for a monad M

(objects) $a \in \mathcal{C}_0$, same objects of $\mathcal{C}$

(arrows) $\mathcal{C}_M[a, b] = \mathcal{C}[a, Mb]$

(identity) on a is η_a

(composition) of $f \in \mathcal{C}_M[a, b]$ and $g \in \mathcal{C}_M[b, c]$ is $g^* \circ f$

Note. The axioms for M amounts to say that $\mathcal{C}_M$ is a category.

Internal Language ML_M for Categories with one Monad

- types $\tau ::= a \mid M\tau$, contexts $\Gamma ::= x{:}\tau$, and their interpretation $[\![M\tau]\!] = M[\![\tau]\!]$
- raw terms $e ::= x \mid f(e) \mid \mathrm{ret}\, e \mid \mathrm{do}\,\{x \leftarrow e_1; e_2\}$, use $\mathrm{ret}_M e$ when more than one M

Additional rules for well-formedness of terms and their interpretation

$$\frac{[\![\Gamma \vdash e{:}\tau]\!] = f}{[\![\Gamma \vdash \mathrm{ret}\, e{:} M\tau]\!] = \eta_a \circ f}\ [\![\tau]\!] = a \qquad \frac{\begin{array}{c}[\![\Gamma \vdash e_1{:} M\tau_1]\!] = f \\ [\![x_1{:}\tau_1 \vdash e_2{:} M\tau_2]\!] = g\end{array}}{[\![\Gamma \vdash \mathrm{do}\,\{x_1 \leftarrow e_1; e_2\}{:} M\tau_2]\!] = g^* \circ f}$$

Additional rules for equality of terms

$$\frac{\Gamma \vdash e{:} M\tau}{\Gamma \vdash \mathrm{do}\,\{x \leftarrow e; \mathrm{ret}\, x\} = e{:} M\tau} \qquad \frac{\Gamma \vdash e_1{:}\tau_1 \quad x_1{:}\tau_1 \vdash e_2{:} M\tau_2}{\Gamma \vdash \mathrm{do}\,\{x_1 \leftarrow \mathrm{ret}\, e_1; e_2\} = e_2[x_1{:} e_1]{:} M\tau_2}$$

$$\frac{\Gamma \vdash e_1{:} M\tau_1 \quad x_1{:}\tau_1 \vdash e_2{:} M\tau_2 \quad x_2{:}\tau_2 \vdash e_3{:} M\tau_3}{\Gamma \vdash \mathrm{do}\,\{x_2 \leftarrow (\mathrm{do}\,\{x_1 \leftarrow e_1; e_2\}); e_3\} = \mathrm{do}\,\{x_1 \leftarrow e_1; \mathrm{do}\,\{x_2 \leftarrow e_2; e_3\}\}{:} M\tau_3}$$

Internal Language $ML_{\times M}$ and Strong Monad on a Cartesian Category

- types $\tau ::= \ldots \mid M\tau$ and contexts $\Gamma ::= 1 \mid \Gamma, x{:}\tau$ with $x \notin \Gamma$
- raw terms $e ::= \ldots \mid \mathrm{ret}\, e \mid \mathrm{do}\,\{x \leftarrow e_1; e_2\}$ with several (occurrences of) variables

Interpretation of $\mathrm{do}\,\{x \leftarrow e_1; e_2\}$ and extension $\dfrac{c \times a \xrightarrow{f} Mb}{c \times Ma \xrightarrow{f^*_c} Mb}$ parametric in c

$$\frac{[\![\Gamma \vdash e_1 : M\tau_1]\!] = f \quad [\![\Gamma, x_1 : \tau_1 \vdash e_2 : M\tau_2]\!] = g}{[\![\Gamma \vdash \mathrm{do}\,\{x_1 \leftarrow e_1; e_2\} : M\tau_2]\!] = g^*_c \circ f} \; c = [\![\Gamma]\!]$$

Prop. If $\mathcal{C}$ has enough points, then f^*_c is uniquely determined by f^*, namely $\forall x : 1 \to c$

$$\begin{array}{ccc} c \times Ma & \xrightarrow{f^*_c} & Mb \\ \uparrow{\scriptstyle x \times \mathrm{id}_{Ma}} & & \uparrow{\scriptstyle g^*} \\ 1 \times Ma & \xrightarrow{\pi_2} & Ma \end{array} \quad \text{when} \quad \begin{array}{ccc} c \times a & \xrightarrow{f} & Mb \\ \uparrow{\scriptstyle x \times \mathrm{id}_a} & & \uparrow{\scriptstyle g} \\ 1 \times a & \xrightarrow{\pi_2} & a \end{array}$$

Internal Language $ML_{\times M}$ and Strong Monad on a Cartesian Category

- types $\tau ::= \ldots \mid M\tau$ and contexts $\Gamma ::= 1 \mid \Gamma, x{:}\tau$ with $x \notin \Gamma$
- raw terms $e ::= \ldots \mid \mathrm{ret}\, e \mid \mathrm{do}\,\{x \leftarrow e_1; e_2\}$ with several (occurrences of) variables

Interpretation of $\mathrm{do}\,\{x \leftarrow e_1; e_2\}$ and extension $\dfrac{c \times a \xrightarrow{f} Mb}{c \times Ma \xrightarrow{f^*_c} Mb}$ parametric in c

$$\frac{[\![\Gamma \vdash e_1 : M\tau_1]\!] = f \quad [\![\Gamma, x_1 : \tau_1 \vdash e_2 : M\tau_2]\!] = g}{[\![\Gamma \vdash \mathrm{do}\,\{x_1 \leftarrow e_1; e_2\} : M\tau_2]\!] = g^*_c \circ f} \; c = [\![\Gamma]\!]$$

Revised rules for equality of terms

$$\frac{\Gamma \vdash e : M\tau}{\Gamma \vdash \mathrm{do}\,\{x \leftarrow e; \mathrm{ret}\, x\} = e : M\tau} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x_1 : \tau_1 \vdash e_2 : M\tau_2}{\Gamma \vdash \mathrm{do}\,\{x_1 \leftarrow \mathrm{ret}\, e_1; e_2\} = e_2[x_1 : e_1] : M\tau_2}$$

$$\frac{\Gamma \vdash e_1 : M\tau_1 \quad \Gamma, x_1 : \tau_1 \vdash e_2 : M\tau_2 \quad \Gamma, x_2 : \tau_2 \vdash e_3 : M\tau_3}{\Gamma \vdash \mathrm{do}\,\{x_2 \leftarrow (\mathrm{do}\,\{x_1 \leftarrow e_1; e_2\}); e_3\} = \mathrm{do}\,\{x_1 \leftarrow e_1; \mathrm{do}\,\{x_2 \leftarrow e_2; e_3\}\} : M\tau_3}$$

Examples of Monads

- If $\mathcal{C}$ is a poset $(X, \leq)$, then a monad M on $\mathcal{C}$ amounts to a monotonic $M\colon X \longrightarrow X$ s.t. $\forall x \in X.x \leq Mx = M(Mx)$

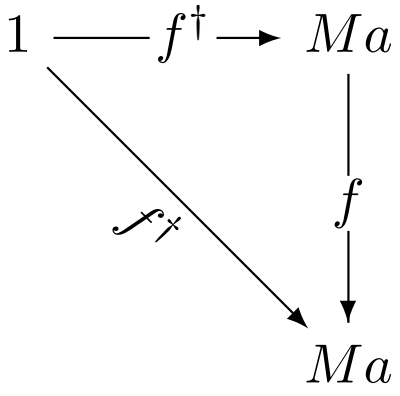
  if $\mathcal{C}$ is also cartesian, i.e. $(X, \leq)$ has finite meets, then M is a strong monad provided M preserves binary meets, i.e. $\forall x, y \in X.M(x \wedge y) = M(x) \wedge M(y)$
- If $(X, \cdot, 1)$ is a monoid, then $MA = A \times X$ is part of a monad on **Set**, and each monoid structure on X induces a different monad
- Go back to the Examples of Computational Types and show that each M is part of a monad (on **Set** or ω**-CPO**)

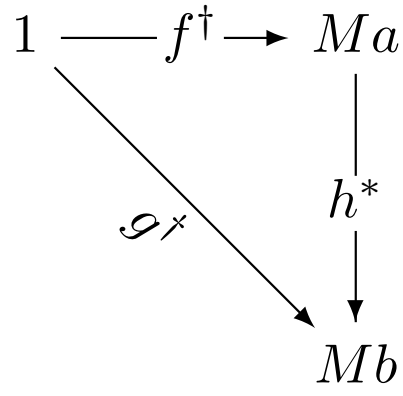
Semantics of Recursive Definitions and $ML_{\times,M}$ with fix

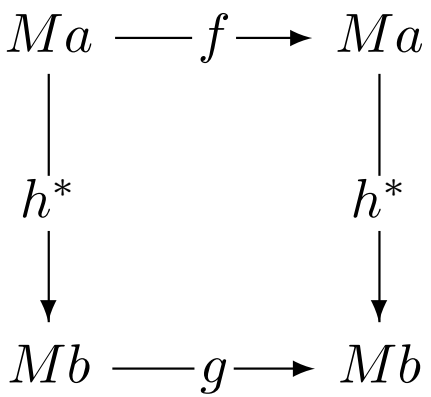
- types $\tau ::= \ldots \mid M\tau$ and contexts $\Gamma ::= 1 \mid \Gamma, x{:}\tau$ with $x \notin \Gamma$
- raw terms $e ::= \ldots \mid \mathrm{ret}\, e \mid \mathrm{do}\,\{x \leftarrow e_1; e_2\} \mid \mathrm{fix}\, x.e$

Interpretation $\dfrac{[\![\Gamma, x{:}M\tau \vdash e{:}M\tau]\!] = f}{[\![\Gamma \vdash \mathrm{fix}\, x.e{:}M\tau]\!] = f^\dagger_c}\; c = [\![\Gamma]\!]$ using $\dfrac{c \times Ma \xrightarrow{f} Ma}{c \xrightarrow{f^\dagger_c} Ma}$

Simplified Semantics $\dfrac{Ma \xrightarrow{f} Ma}{1 \xrightarrow{f^\dagger} Ma}$ in a Cartesian Category with a Monad

- 



when


and $a \xrightarrow{h} Mb$

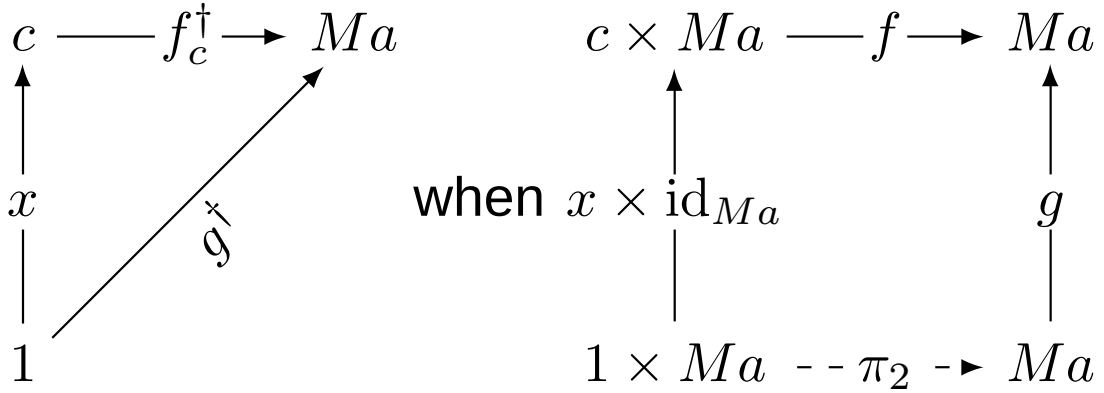
Example. The least fix-point in the category of ω-**CPO** for the lifting monad $MA = A_\perp$.

Semantics of Recursive Definitions and $ML_{\times,M}$ with fix

- types $\tau ::= \ldots \mid M\tau$ and contexts $\Gamma ::= 1 \mid \Gamma, x{:}\tau$ with $x \notin \Gamma$
- raw terms $e ::= \ldots \mid \mathrm{ret}\, e \mid \mathrm{do}\,\{x \leftarrow e_1; e_2\} \mid \mathrm{fix}\, x.e$

Interpretation $\dfrac{[\![\Gamma, x{:}M\tau \vdash e{:}M\tau]\!] = f}{[\![\Gamma \vdash \mathrm{fix}\, x.e{:}M\tau]\!] = f^\dagger_c}\ c = [\![\Gamma]\!]$ using $\dfrac{c \times Ma \xrightarrow{f} Ma}{c \xrightarrow{f^\dagger_c} Ma}$

Prop. If $\mathcal{C}$ has enough points, then $f^\dagger_c$ is uniquely determined by $f^\dagger$, namely $\forall x{:}1 \to c$



Semantics of Recursive Definitions and $ML_{\times,M}$ with fix

- types $\tau ::= \ldots \mid M\tau$ and contexts $\Gamma ::= 1 \mid \Gamma, x{:}\tau$ with $x \notin \Gamma$
- raw terms $e ::= \ldots \mid \mathrm{ret}\, e \mid \mathrm{do}\,\{x \leftarrow e_1; e_2\} \mid \mathrm{fix}\, x.e$

Interpretation $\dfrac{[\![\Gamma, x{:}M\tau \vdash e{:}M\tau]\!] = f}{[\![\Gamma \vdash \mathrm{fix}\, x.e{:}M\tau]\!] = f^\dagger_c}\ c = [\![\Gamma]\!]$ using $\dfrac{c \times Ma \xrightarrow{f} Ma}{c \xrightarrow{f^\dagger_c} Ma}$

Revised rules for equality of terms

- fix-point $\dfrac{\Gamma, x{:}M\tau \vdash e{:}M\tau}{\Gamma \vdash \mathrm{fix}\, x.e = e[x{:}\mathrm{fix}\, x.e]{:}M\tau}$
- uniformity $\dfrac{\Gamma, x_1{:}M\tau_1 \vdash e_1{:}M\tau_1 \quad \Gamma, x_2{:}M\tau_2 \vdash e_2{:}M\tau_2 \quad \Gamma, x{:}\tau_1 \vdash e{:}M\tau_2 \quad \Gamma, x_1{:}M\tau_1 \vdash e_2[x_2{:}\mathrm{do}\,\{x \leftarrow e_1; e\}] = \mathrm{do}\,\{x \leftarrow e_1; e\}{:}M\tau_2}{\Gamma \vdash \mathrm{fix}\, x_2.e_2 = \mathrm{do}\,\{x \leftarrow (\mathrm{fix}\, x_1.e_1); e\}{:}M\tau_2}$

[Asperti-Longo 1991, Ch 3]

Functors

A *functor* F from $\mathcal{C}$ to $\mathcal{D}$, notation $\boxed{F\colon \mathcal{C} \longrightarrow \mathcal{D}}$, consists of

- operations $F_0\colon \mathcal{C}_0 \longrightarrow \mathcal{D}_0$ and $F_1\colon \mathcal{C}_1 \longrightarrow \mathcal{D}_1$ $\boxed{\text{subscripts are usually omitted}}$ s.t.
- F preserves domain and codomain: $\boxed{a \xrightarrow{f} b \text{ in } \mathcal{C} \text{ implies } Fa \xrightarrow{Ff} Fb \text{ in } \mathcal{D}}$
- F preserves identity and composition: $\boxed{F(\mathrm{id}_a) = \mathrm{id}_{Fa} \text{ and } F(g \circ f) = Fg \circ Ff}$

Cat is the category of (small) categories and functors (the definition of identity functors and functor composition are obvious).

Functors

Dogma 2 [Gog91]: to any *natural* construction on structures of one species, yielding structures of another species, there corresponds a **functor**.

- Functors between discrete categories correspond to functions between the underlying collections of objects
- Functors between preorders correspond to monotonic maps
- Functors between monoids correspond to monoid homomorphisms
- Given $\mathcal{C}$ whose objects are sets with *additional structure* (and arrows are functions *respecting* the structure), there is a *forgetful functor* $U : \mathcal{C} \longrightarrow$ **Set**, which maps an object to the underlying set and is the identity on arrows (thus U is faithful). Examples are: **Mon**, **PO**, ω**-CPO**, **Alg**$_\Sigma$, **EN**, $\underline{A}$**-Set**. Similarly one can define
- $U_0, U_1 :$ **Graph** $\longrightarrow$ **Set** mapping a graph to the underlying set of nodes/arcs.
- $U :$ **Cat** $\longrightarrow$ **Graph** mapping a category to the underlying graph.

Functors

Dogma 2 [Gog91]: to any *natural* construction on structures of one species, yielding structures of another species, there corresponds a **functor**.

- *diagonal functor* $\Delta\colon \mathcal{C} \longrightarrow \mathcal{C} \times \mathcal{C}$ is given by $\Delta(a) = (a, a)$ and $\Delta(f) = (f, f)$
- *projection functor* $\pi_i\colon \mathcal{C}_1 \times \mathcal{C}_2 \longrightarrow \mathcal{C}_i$ is given by $\pi_i(a_1, a_2) = a_i$ and $\pi_i(f_1, f_2) = f_i$
- Given a biCC $\mathcal{C}$, we define the following functors (using choice)
  - $\times\colon \mathcal{C} \times \mathcal{C} \longrightarrow \mathcal{C}$ mapping (a_1, a_2) to $a_1 \times a_2$, where $a_1 \xleftarrow{\pi_1} a_1 \times a_2 \xrightarrow{\pi_2} a_2$ is a *chosen* product diagram
  - $+\colon \mathcal{C} \times \mathcal{C} \longrightarrow \mathcal{C}$ mapping (a_1, a_2) to $a_1 + a_2$, where $a_1 \xrightarrow{\iota_1} a_1 + a_2 \xleftarrow{\iota_2} a_2$ is a *chosen* coproduct diagram

  The definition of $f_1 \times f_2$, $f_1 + f_2$ (and the proof that they preserve identities and composition) exploit the universal properties of products and coproducts.
- Give examples of construction on sets that do not extend to a functor on **Set**

  e.g. case analysis on the cardinality of X set, or on whether X is a member of a given set.
- Give more examples of functors between categories defined so far

  inclusion functor from **EN** into $\underline{A}$**-Set**, exploiting the encoding of N in any non-trivial pCA

  inclusion functor from **Set** into ω**-CPO**, **PO** and $\underline{A}$**-Set**

Natural Transformations

Given two functors $F, G : \mathcal{C} \longrightarrow \mathcal{D}$, a *natural transformation* $\boxed{\tau : F \longrightarrow G}$ consists of an operation $\tau : \mathcal{C}_0 \longrightarrow \mathcal{D}_1$, we may write $\boxed{\tau_a \text{ for } \tau(a)}$, s.t.

$\boxed{\forall a \in \mathcal{C}. \tau_a \in \mathcal{D}[Fa, Ga]}$ and $\boxed{\forall a, b \in \mathcal{C}. \forall f \in \mathcal{C}[a, b].\ \tau_b \circ Ff = Gf \circ \tau_a}$

or equivalently, for all $a \xrightarrow{f} b$ in $\mathcal{C}$ the square[a]

$$\begin{array}{ccc} Fa & \xrightarrow{\tau_a} & Ga \\ \downarrow{\scriptstyle Ff} & & \downarrow{\scriptstyle Gf} \\ Ga & \xrightarrow{\tau_b} & Gb \end{array}$$

commutes in $\mathcal{D}$

To make explicit also the categories involved we write $\boxed{\mathcal{C} \overset{F}{\underset{G}{\Downarrow \tau}} \mathcal{D}}$

Natural Transformations

Dogma 3: to each *natural translation* from a construction $F\colon \mathcal{A} \longrightarrow \mathcal{B}$ to a construction $G\colon \mathcal{A} \longrightarrow \mathcal{B}$ there corresponds a **natural transformation** $F \longrightarrow G$.

- the *identity natural transformation* $\mathcal{A} \overset{\longrightarrow F \longrightarrow}{\Downarrow \mathrm{id}_F} \mathcal{B}$ is $\mathrm{id}_F(a) = \mathrm{id}_{Fa}$
- if $\mathcal{A} \overset{\longrightarrow F_1 \longrightarrow}{\Downarrow \tau_1}\overset{\longrightarrow F_2 \longrightarrow}{\Downarrow \tau_2}\overset{\longrightarrow F_3 \longrightarrow}{} \mathcal{B}$ the *vertical composite* $\mathcal{A} \overset{\longrightarrow F_1 \longrightarrow}{\Downarrow \tau_2 \circ \tau_1} \mathcal{B}$ is $(\tau_2 \circ \tau_1)_a = \tau_2(a) \circ \tau_1(a)$

In fact, (when $\mathcal{A}$ and $\mathcal{B}$ are small) there is a *functor category* $\mathcal{B}^{\mathcal{A}}$ of functors $F\colon \mathcal{A} \longrightarrow \mathcal{B}$ and natural transformations, which is an exponential object in **Cat**.

- Universal properties induce both functors and natural transformations, e.g. if $\mathcal{C}$ is a biCC, then in addition to the functors $- \times -$ and $- + -$ we have

  $\mathcal{C} \times \mathcal{C} \overset{\longrightarrow \times \longrightarrow}{\Downarrow \pi_i}\overset{\longrightarrow \pi_i \longrightarrow}{} \mathcal{C}$ $\quad$ $\mathcal{C} \times \mathcal{C} \overset{\longrightarrow \pi_i \longrightarrow}{\Downarrow \iota_i}\overset{\longrightarrow + \longrightarrow}{} \mathcal{C}$ $\quad$ where

  $\pi_i(a_1, a_2)$ is $a_1 \times a_2 \xrightarrow{\pi_i} a_i$, $\iota_i(a_1, a_2)$ is $a_i \xrightarrow{\iota_i} a_1 + a_2$

Monad M on $\mathcal{C}$ in monoid form (M, η, μ)

- $M : \mathcal{C} \longrightarrow \mathcal{C}$ functor $\mathcal{C} \xrightarrow[M]{\overset{\mathrm{id}_{\mathcal{C}}}{\Downarrow \eta}} \mathcal{C}$ $\mathcal{C} \xrightarrow[M]{\overset{M^2}{\Downarrow \mu}} \mathcal{C}$ natural transformations s.t.

- $$\begin{array}{ccc} M^3a & \xrightarrow{\mu_{Ma}} & M^2a \\ {\scriptstyle M\mu_a}\downarrow & & \downarrow{\scriptstyle \mu_a} \\ M^2a & \xrightarrow[\mu_a]{} & Ma \end{array} \qquad \begin{array}{ccccc} Ma & \xrightarrow{\eta_{Ma}} & M^2a & \xleftarrow{M\eta_a} & Ma \\ & {\scriptstyle \mathrm{id}_{Ma}}\searrow & \downarrow{\scriptstyle \mu_a} & \swarrow{\scriptstyle \mathrm{id}_{Ma}} & \\ & & Ma & & \end{array}$$

Prop. There is a bijection monads in monoid form and extension form

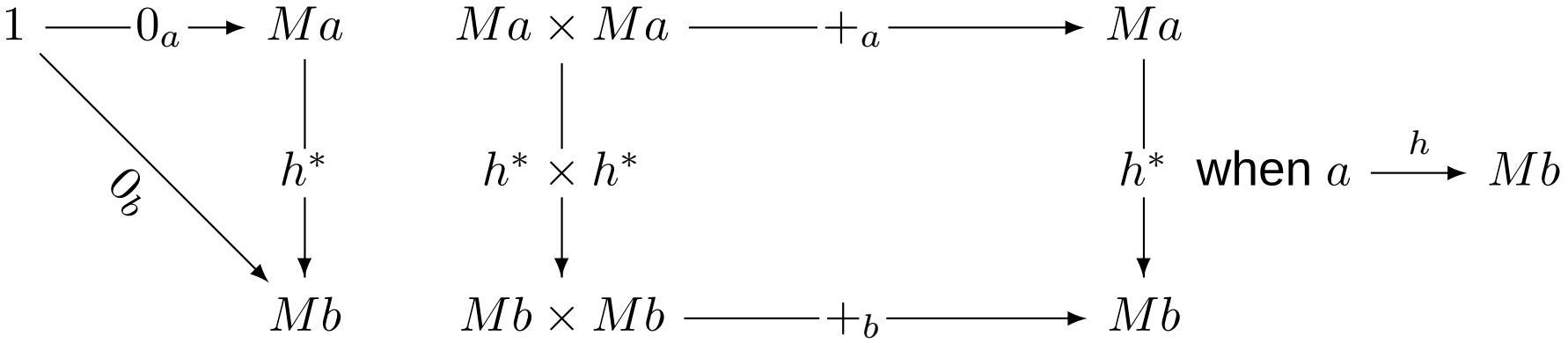
- $(f : a \longrightarrow Mb)^* \triangleq Ma \xrightarrow{Mf} M^2b \xrightarrow{\mu_b} Mb$
- $M(f : a \longrightarrow b) \triangleq (a \xrightarrow{f} b \xrightarrow{\eta_b} Mb)^*$
- $\mu_a \triangleq (Ma \xrightarrow{\mathrm{id}_{Ma}} Ma)^*$

Note. To verify that "(M, η, μ) is a monad" there are 7 equations to check!

Semantics of Collection Types and $ML_{\times,M}$ with 0 and $+$

- types $\tau ::= \ldots \mid M\tau$ and contexts $\Gamma ::= 1 \mid \Gamma, x{:}\tau$ with $x \notin \Gamma$
- raw terms $e ::= \ldots \mid \{e\} \mid \mathrm{flat}\, e \mid \{e_2 | x \leftarrow e_1\} \mid 0 \mid \ \mid e_1 + e_2$
- translation $\{e\} \triangleq \mathrm{ret}\, e \quad \mathrm{flat}\, e \triangleq \mathrm{do}\,\{x \leftarrow e; x\} \quad \{e_2 | x \leftarrow e_1\} \triangleq \mathrm{do}\,\{x \leftarrow e_1; \mathrm{ret}\, e_2\}$

Algebraic operations $1 \xrightarrow{0_a} Ma$ and $Ma \times Ma \xrightarrow{+_a} Ma$ and semantics of 0 and $+$

- 

when $a \xrightarrow{h} Mb$

Semantics of Collection Types and $ML_{\times,M}$ with 0 and $+$

- types $\tau ::= \ldots \mid M\tau$ and contexts $\Gamma ::= 1 \mid \Gamma, x{:}\tau$ with $x \notin \Gamma$
- raw terms $e ::= \ldots \mid \{e\} \mid \mathrm{flat}\, e \mid \{e_2 | x \leftarrow e_1\} \mid 0 \mid \ \mid e_1 + e_2$
- translation $\{e\} \stackrel{\Delta}{=} \mathrm{ret}\, e \quad \mathrm{flat}\, e \stackrel{\Delta}{=} \mathrm{do}\,\{x \leftarrow e; x\} \quad \{e_2 | x \leftarrow e_1\} \stackrel{\Delta}{=} \mathrm{do}\,\{x \leftarrow e_1; \mathrm{ret}\, e_2\}$

Revised rules for equality of terms

- algebraicity $$\frac{\Gamma, x{:}\tau_1 \vdash e{:}M\tau_2}{\Gamma \vdash \mathrm{do}\,\{x \leftarrow 0; e\} = 0{:}M\tau_2}$$

$$\frac{\Gamma \vdash e_1, e_2{:}M\tau_1 \quad \Gamma, x{:}\tau_1 \vdash e{:}M\tau_2}{\Gamma \vdash \mathrm{do}\,\{x \leftarrow (e_1 + e_2); e\} = (\mathrm{do}\,\{x \leftarrow e_1; e\}) + (\mathrm{do}\,\{x \leftarrow e_2; e\}){:}M\tau_2}$$

- further properties of 0 and $+$: they form a (commutative, idempotent) monoid.

Examples. monads $MA = T_\Sigma(A)/=_E$, where Σ has a constant 0 and a binary operation $+$, and $=_E$ is the theory induced by a set E of equations on Σ-terms.

PART 3

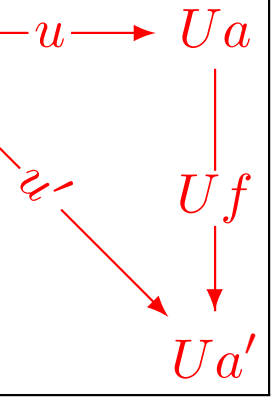
Algebra and Monads
[Asperti-Longo 1991, Ch 5] and [Manes 1976, Man98]

Universal Arrows

Given a functor $U : \mathcal{A} \longrightarrow \mathcal{C}$ and an object $c \in \mathcal{C}$

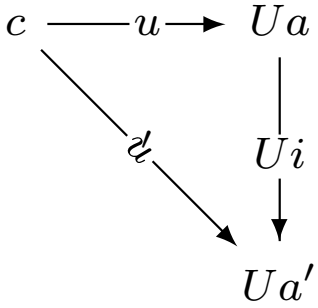
- a *universal arrow* from c to U consists of a pair (u, a) with $a \in \mathcal{A}$ and $c \xrightarrow{u} Ua$ s.t.

$\forall a' \in \mathcal{A}. \forall u' : c \longrightarrow Ua'.\ \exists! f : a \longrightarrow a'$ s.t.



- a universal arrow (u, a) from c to U is determined up to unique iso
  - if $a \xrightarrow{i} a'$ is an iso in $\mathcal{A}$, then $((Ui) \circ u, a')$ is a universal arrow from c to U
  - if (u', a') is a universal arrow from c to U, then

$\exists! i : a \longrightarrow a'$ iso s.t.



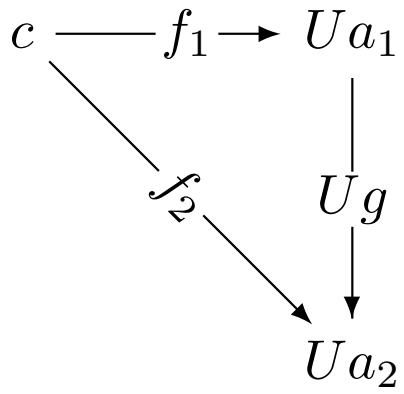
- there is a dual notion of *universal arrow* from U to c.

Reformulations and Examples

A universal arrow (u, a) from $c \in \mathcal{C}$ to $U\colon \mathcal{A} \longrightarrow \mathcal{C}$ corresponds to an initial object in the category $c \uparrow U$ given by

(objects) (f, a) with $a \in \mathcal{A}$ and $f \in \mathcal{C}[c, Ua]$

(arrows) $(f_1, a_1) \xrightarrow{g} (f_2, a_2)$ with $g \in \mathcal{A}[a_1, a_2]$ s.t.



commutes

Reformulations and Examples

Any universal property (for $\mathcal{A}$) introduced so far can be recast in terms of universal arrows to/from a functor $U\colon \mathcal{A} \longrightarrow \mathcal{C}$ by a suitable choice of $\mathcal{C}$ and U, for instance

- an I-indexed coproduct diagram $\iota_i\colon a_i \longrightarrow a$ corresponds to a universal arrow $((\iota_i|i \in I), a)$ from $(a_i|i \in I)$ to $\Delta\colon \mathcal{A} \longrightarrow \mathcal{A}^I$, where $\Delta(a) \stackrel{\Delta}{=} (a|i \in I)$ and $\mathcal{A}^I$ is

  (objects) I-indexed families $a = (a_i|i \in I)$ of objects of $\mathcal{A}$

  (arrows) $(f_i|i \in I)\colon a \longrightarrow b$ provided $\forall i \in I.f_i \in \mathcal{A}[a_i, b_i]$

  dually, I-indexed product diagrams corresponds to universal arrows from Δ

When $U\colon \mathcal{A} \longrightarrow \mathcal{C}$ is a monotonic maps between posets, a universal arrow from c to U amounts to *the least* a s.t. $c \leq Ua$.

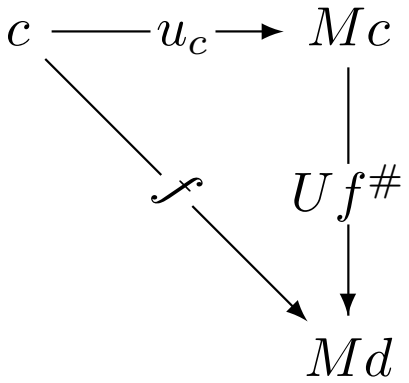
We have given several *forgetful* functors $U\colon \mathcal{A} \longrightarrow$ **Set**, do the universal arrows to/from these functors exists?

$\mathcal{A}$	a_X s.t. $u\colon X \longrightarrow U(a_X)$ univ.	b_X s.t. $u\colon U(b_X) \longrightarrow X$ univ.
Mon	$a_X =$ free monoid X^* on X	when $\lvert X\rvert = 1$: $b_X = 1$
PO	$a_X = (X, =)$ discrete p.o. on X	when $\lvert X\rvert \leq 1$: $b_X = (X, =)$
Alg$_\Sigma$	$a_X =$ free Σ-algebra $T_\Sigma(X)$ on X	NO unless Σ trivial

Universal Arrows and Monads

Prop. Given a functor $U\colon \mathcal{A} \longrightarrow \mathcal{C}$ and a universal arrow (u_c, a_c) from c to U for every $c \in \mathcal{C}$, one has a monad $(M, \eta, -^*)$ on $\mathcal{C}$ given by

- $Mc \triangleq U(a_c)$
- $\eta_c \triangleq u_c\colon c \longrightarrow Mc$
- $f^* \triangleq U(f^\#)\colon Mc \longrightarrow Md$, with $f^\#$ unique arrow in $\mathcal{A}[a_c, a_d]$ s.t.



Prop Every monad M on $\mathcal{C}$ is obtained in the way described above, by a suitable choice of $\mathcal{A}$ and $U\colon \mathcal{A} \longrightarrow \mathcal{C}$. In fact, there are two *canonical* choices for $\mathcal{A}$ (and U):

- The Eilenberg-Moore category $\mathcal{C}^M$ of *EM-algebra*
- The Kleisli category $\mathcal{C}_M$ of *programs* (already introduced)

Universal Arrows and Monads

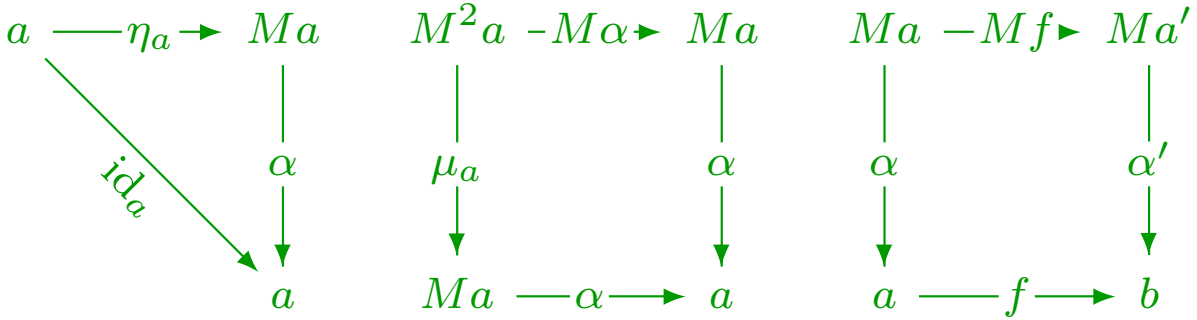
Def. The Eilenberg-Moore category $\mathcal{C}^M$ of EM-algebra

(objects) $\underline{a} = (a \in \mathcal{C}_0, \alpha \in \mathcal{C}[Ma, a])$ s.t.

$x{:}\, a \vdash \alpha(\{x\}) = x{:}\, a$ and $x_2{:}\, M^2 a \vdash \alpha(\mathrm{flat}\, x_2) = \alpha(\{\alpha(x_1) | x_1 \leftarrow x_2\}){:}\, Ma$

(arrows) $\mathcal{C}^M[\underline{a}, \underline{a'}]$ are $f \in \mathcal{C}[a, a']$ s.t. $x_1{:}\, Ma \vdash f(\alpha(x_1)) = \alpha'(\{f(x) | x \leftarrow x_1\}){:}\, a'$

commuting diagrams equivalent to the equations above



(identity) on $\underline{a}$ is id_a

(composition) of $f \in \mathcal{C}^M[\underline{a}_1, \underline{a}_2]$ and $g \in \mathcal{C}^M[\underline{a}_2, \underline{a}_3]$ is $g \circ f$

The forgetful functor $U{:}\, \mathcal{C}^M \longrightarrow \mathcal{C}$

- $U(\underline{a}) \stackrel{\Delta}{=} a$
- $U(f) \stackrel{\Delta}{=} f \in \mathcal{C}[a, b]$ when $f \in \mathcal{C}^M[\underline{a}, \underline{b}]$

Universal Arrows and Monads

Def. The Klesli category $\mathcal{C}_M$ of programs

(objects) $a \in \mathcal{C}_0$, same objects of $\mathcal{C}$

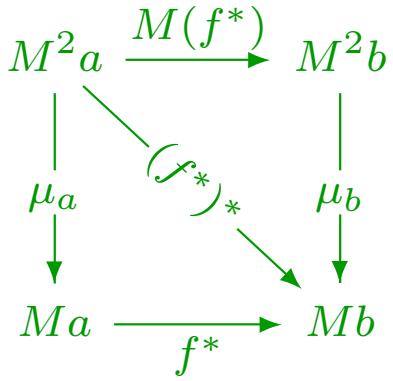
(arrows) $\mathcal{C}_M[a,b] = \mathcal{C}[a, Mb]$

(identity) on a is η_a

(composition) of $f \in \mathcal{C}_M[a,b]$ and $g \in \mathcal{C}_M[b,c]$ is $g^* \circ f$

The forgetful functor $U : \mathcal{C}_M \longrightarrow \mathcal{C}$ and the *comparision functor* $F : \mathcal{C}_M \longrightarrow \mathcal{C}^M$

- $U(a) \triangleq Ma$
- $U(f) \triangleq f^* \in \mathcal{C}[Ma, Mb]$ when $f \in \mathcal{C}_M[a,b]$
- $F(a) \triangleq (Ma, \mu_a)$ the *free* EM-algebra on a
- $F(f) \triangleq f^* \in \mathcal{C}^M[Fa, Fb]$ when $f \in \mathcal{C}_M[a,b]$, because



Algebraic Theories and Monads on $\mathcal{C} = \mathbf{Set}$

Def. An **algebraic theory** is a pair (Σ, E), with Σ signature Σ and E set of Σ-equations (with variables in nat). (Σ, E) induces a monad M on $\mathcal{C}$, where $MA = T_\Sigma(A)/=_E$ is the set of Σ-terms with variables in A modulo the congruence $=_E$ induced by E

- $\mathcal{C}^M$ is (isomorphic to) the category **Alg**$_{(\Sigma,E)}$ of Σ-algebra satisfying equations E
- $\mathcal{C}_M$ is (isomorphic to) the dual of $T_{(\Sigma,E)}$, the category of sets and substitutions modulo $=_E$, and the comparison functor is $F(X) = T_\Sigma(X)/=_E$.

Def. A functor/monad M on $\mathcal{C}$ is **finitary** $\stackrel{\Delta}{\Longleftrightarrow} \forall X \in \mathcal{C}_0. \forall y \in MX.$ exist $X_0 \subseteq_{fin} X$ and $y_0 \in MX_0$ s.t. $Mi: y_0 \longmapsto y$, where $i: X_0 \longrightarrow X$ is the inclusion map of X_0 into X. The (finite) set X_0 is called a **support** of y.

Prop. Finitary monads are (modulo iso) the monads induced by algebraic theories.

Prop. There is a bijection from $\mathrm{op}' \in Mn$ to n-ary **algebraic operations** op for M, i.e. $\mathrm{op}_a: (Ma)^n \longrightarrow Ma$ s.t. $h^*(\mathrm{op}_a(t_i | i \in n)) = \mathrm{op}_b(h^*(t_i) | i \in n)$ when $a \stackrel{h}{\longrightarrow} Mb$.
Proof. $\mathrm{op}' = \mathrm{op}_n(\mathrm{ret}\, i | i \in n)$ and $\mathrm{op}_a(t) = \mathrm{do}\,\{i \leftarrow \mathrm{op}'; t(i)\}$.

The bijective correspondence holds for any strong monad M on a CCC $\mathcal{C}$.

Algebraic Theories and Monads on $\mathcal{C} = \mathbf{Set}$

Def. [Man98] M is a **collection** functor/**monad** $\overset{\Delta}{\Longleftrightarrow}$

- M finitary and for any $X \in \mathcal{C}$ and $y \in MX$ exists the minimum support $\sigma_X(y)$ of y
- and $\sigma_X : MX \longrightarrow \mathcal{P}_{fin}(X)$ is a natural transformation/**monad morphism**, i.e.
  - $x : X \vdash \sigma_X(\mathrm{ret}\, x) = \{x\} : \mathcal{P}_{fin}(X)$
  - $c : MX \vdash \sigma_X(\mathrm{do}\,\{x \leftarrow c; f(x)\}) = \{x | x \in \sigma_X(c) \wedge y \in \sigma_Y(f(x))\} : \mathcal{P}_{fin}(Y)$

Prop. Collection monads are the monads induced by **balanced** algebraic theories (Σ, E), i.e. $\mathrm{FV}(t) = \mathrm{FV}(t')$ for any equation $t = t'$ in E.
Examples. See [Man98] for examples and counter-examples of collection monads.

Algebraic Semantics of Collection Types in **Set**

- a collection monad M (membership of a collection is well-defined)
- $0' \in M0$ and $+' \in M2$ (empty collection and union of collections)
- *aggregate operations* $\mathrm{op}_X : MX \longrightarrow A$ as *homomorphic extension* $f^{\#} : (MX, \mu_X) \longrightarrow \underline{A}$ of $f : X \longrightarrow A$ with $\underline{A} = (A, \alpha)$ EM-algebra, i.e. defined by *structural induction* on collections.

Algebraic Theories and Monads on $\mathcal{C} = \mathbf{Set}$

Def. [Man98] M is a **collection** functor/**monad** $\overset{\Delta}{\Longleftrightarrow}$

- M finitary and for any $X \in \mathcal{C}$ and $y \in MX$ exists the minimum support $\sigma_X(y)$ of y
- and $\sigma_X : MX \longrightarrow \mathcal{P}_{fin}(X)$ is a natural transformation/**monad morphism**, i.e.
  - $x : X \vdash \sigma_X(\mathrm{ret}\, x) = \{x\} : \mathcal{P}_{fin}(X)$
  - $c : MX \vdash \sigma_X(\mathrm{do}\,\{x \leftarrow c; f(x)\}) = \{x | x \in \sigma_X(c) \wedge y \in \sigma_Y(f(x))\} : \mathcal{P}_{fin}(Y)$

Prop. Collection monads are the monads induced by **balanced** algebraic theories (Σ, E), i.e. $\mathrm{FV}(t) = \mathrm{FV}(t')$ for any equation $t = t'$ in E.

Examples of EM-algebra for collection Monads

- for finite sets $\exists, \forall : \mathcal{P}_{fin}(\mathrm{bool}) \longrightarrow \mathrm{bool}$, $\max : \mathcal{P}_{fin}(\mathrm{nat}) \longrightarrow \mathrm{nat}$
- for finite bags $\sum, \prod : (\mathrm{nat} \overset{fin}{\to} \mathrm{nat}) \longrightarrow \mathrm{nat}$
- for finite lists $cat : \mathrm{string}^* \longrightarrow \mathrm{string}$

Algebraic approach to Computational Effects [PP01,HPP02,PP09]

Basic idea: computational types $M\tau$ as (abstract) datatypes in a biCCC $\mathcal{C}$, with operations $\mathrm{op}_x : H(Mx) \stackrel{\cdot}{\to} Mx$ to build computations

1. is op *algebraic*, i.e. definable from some $\mathrm{op}'_x : Hx \stackrel{\cdot}{\to} Mx$ or $\mathrm{op}' : b \longrightarrow Ma$?
2. is M the monad induced by an *algebraic theory*? M is no longer *abstract*!

   There are 3 ways of combining algebraic theories (computational effects)

   sum $\Sigma + \Sigma'$ and $E + E'$ (disjoint union of operations and equations)

   tensor sum + $\mathrm{op}'(\mathrm{op}(x_{i,j}|i \in m)|j \in n) = \mathrm{op}(\mathrm{op}'(x_{i,j}|j \in n)|i \in m)$

   distribute sum + $\mathrm{op}'(\overline{x}, \mathrm{op}(x_i|i \in m), \overline{x}') = \mathrm{op}(\mathrm{op}'(\overline{x}, x_i, \overline{x}'|i \in m))$
3. is op an *effect handler*, i.e. definable as *homomorphic extension* $f^{\#} : Mx \longrightarrow a$ for some EM-algebra $\underline{a}$ and $f : x \longrightarrow a$?

Examples of Algebraic Operations

- side-effects $lookup : L \longrightarrow MU$, $update : L \times U \longrightarrow M1$, $new : U \longrightarrow ML$
- input-output $read : 1 \longrightarrow MU$, $write : U \longrightarrow M1$
- exceptions $raise : E \longrightarrow M0$
- continuations $abort : R \longrightarrow MR$, $callcc_X : (M0)^{((M0)^X)} \longrightarrow MX$

Algebraic approach to Computational Effects [PP01,HPP02,PP09]

Basic idea: computational types $M\tau$ as (abstract) datatypes in a biCCC $\mathcal{C}$, with operations $\mathrm{op}_x : H(Mx) \stackrel{\cdot}{\to} Mx$ to build computations

1. is op *algebraic*, i.e. definable from some $\mathrm{op}'_x : Hx \stackrel{\cdot}{\to} Mx$ or $\mathrm{op}' : b \longrightarrow Ma$?
2. is M the monad induced by an *algebraic theory*? M is no longer *abstract*!

   There are 3 ways of combining algebraic theories (computational effects)

   sum $\Sigma + \Sigma'$ and $E + E'$ (disjoint union of operations and equations)

   tensor sum + $\mathrm{op}'(\mathrm{op}(x_{i,j}|i \in m)|j \in n) = \mathrm{op}(\mathrm{op}'(x_{i,j}|j \in n)|i \in m)$

   distribute sum + $\mathrm{op}'(\overline{x}, \mathrm{op}(x_i|i \in m), \overline{x}') = \mathrm{op}(\mathrm{op}'(\overline{x}, x_i, \overline{x}'|i \in m))$
3. is op an *effect handler*, i.e. definable as *homomorphic extension* $f^{\#} : Mx \longrightarrow a$ for some EM-algebra $\underline{a}$ and $f : x \longrightarrow a$?

Examples of Non-Algebraic Operations

- state-readers $local : S \times MX \longrightarrow MX$ where $local(s, c) = \lambda s'.c(s)$
- exceptions $handle : MX \times (MX)^E \longrightarrow MX$ where $handle(\mathrm{ret}\, x, h) = \mathrm{ret}\, x$ and $handle(raise(e), h) = h(e)$
- nondeterministic interactive programs $por : MX \times MX \longrightarrow MX$

PART 4

Monads in Haskell

Some Type Classes in Haskell

Type class = type (constructor) equipped with some (polymorphic) operations, but there is no way to ensure that the operations satisfy certain (equational) properties

- class Eq a where
  $==:: a \to a \to$ Bool
  a type a equipped with a test for equality
- class Functor f where
  fmap: : $(a \to b) \to f\ a \to f\ b$
  a type constructor f with the structure of a (strong) endofuntor
- class Monad m where
  $>>=:: m\ a \to (a \to m\ b) \to m\ b$
  ret: : $a \to ma$
  a type constructor m with the structure of a (strong) monad
- class (Monad m)$\Rightarrow$MonadPlus m where
  mplus: : $m\ a \to m\ a \to m\ a$
  mzero: : ma
  a (strong) monad with union and empty collection operations

The IO Monad and Interaction with the World

- $IO\ a$ abstract datatype, whose values describe interactions with the *world*
- $\mathrm{op}_x : H(IO\ x) \to IO\ x$ (polymorphic) operation that may have an effect, e.g. read/write operations on a file
- the IO-interpreter runs at the top level, it interprets only the IO-operations, and relies on the *pure interpreter* $e \to v$ for the rest
- IO-computations may spawn parallel threads, thus the pure interpreter should cope with concurrent calls!

  Parallelism is not a *problem* in a pure functional language, when implementations are based on term graph rewriting with in-place update.

The ST_s Monad and Encapsulation of State

- $ST_s a$ abstract datatype for imperative computations within *region* s
  in Haskell a region (variable) is *represented* by a type (variable)
  - $STRef_s a$ (abstract) datatype of references (to values of type a) in region s
  - $newSTRef : a \to ST_s(STRef_s a)$
  - $readSTRef : STRef_s a \to ST_s a$
  - $writeSTRef : STRef_s a \to a \to ST_s()$
  - $eqSTRef : STRef_s a \to STRef_s a \to \mathrm{bool}$

The ST_s Monad and Encapsulation of State

- $ST_s a$ abstract datatype for imperative computations within *region* s
- the ST-interpreter (for region s) interprets only the ST-operations, and relies on the *pure interpreter* $e \to v$ for the rest
- the pure interpreter may activate the ST-interpreter using the operation $runST\colon (\forall s.ST_s a) \to a$, pure evaluation of $(runST\ e)\colon \tau$ involves
  - a call to the ST-interpreter, which evaluates $e_s\colon ST_s \tau$ in a *fresh* region s
  - upon completion a value $v\colon \tau$ is returned to the pure interpreter, and the state in region s is erased (region based memory management)

    in any reasonable implementation, after evaluation $runST\ e$ is replaced by v.
- the Type System ensures that it is safe to reclaim the store at the end of the imperative computation within region s, and that imperative computations in different regions do not interfere (thus parallelism is not a problem).

$$IO \longrightarrow \text{pure} \underset{\longleftarrow}{\overset{runST}{\longrightarrow}} ST$$

  - IO: global side-effects, concurrency/interference
  - pure: no side-effects, confluence (concurrency without interference)
  - ST_s: local side-effects, single-threaded (no concurrency)

Bibliographic References

On the benefits of Category Theory

Goguen:91 A Categorical Manifesto. MSCS 1(1), 1991

Scott:80 Relating theories of the lambda-calculus. In To H.B. Curry: Essays in Combinatory Logic, Lambda calculus and formalism. Hindley R., Seldin J. (ed.), 1980

Computational Types and Monads

Moggi:88 Computational lambda-calculus and monads. Edinburgh Univ. ECS-LFCS-88-66, 1988

Moggi:89 Computational lambda-calculus and monads. LICS, 1989

Moggi:91 Notions of computation and monads. I&C 93(1), 1991

Moggi:97 Metalanguages and applications. In Semantics and Logics of Computation, CUP, 1997

Benton+Hughes+Moggi:02 Monads and Effects. In APPSEM 2000, LNCS 2002

Jaskelioff:09 Modular monad transformers. ESOP, 2009

Fixpoint semantics

Crole+Pitts:90 New foundations for fixpoint computations. LICS,

Evaluation Logic

Pitts:91 Evaluation logic. Banff Workshop, 1990

Moggi:94 A general semantics for evaluation logic. LICS, 1994

Monads and (Fuctional) Programming

Wadler:90 Comprehending monads. ICFP, 199

Wadler:92 The essence of functional programming. POPL, 1992

Wadler+Thiemann:03 The marriage of effects and monads. ACM Trans. on Comp. Logic 4, 2003

Lucassen+Gifford:88 Polymorphic Effect Systems. POPL, 1987

Collection Types and Monads

Buneman+Naqvi+Tannen+Wong:95 Principles of programming with complex objects and collection types. TCS 149(1), 1995

Manes:76 Algebraic Theories, Ernest G. Manes. Graduate Texts in Mathematics (Springer), 1976

Manes:98 Implementing collection classes with monads. MSCS 8(3), 1998

Monads in Haskell

PeytonJones+Wadler:93 Imperative functional programming. POPL, 1993

Launchbury+PeytonJones:95 State in Haskell. Lisp&Symb. Comp. 8(4), 1995

Harris+Marlow+PeytonJones+Herlihy:05 Composable memory transactions. PPoPP, 2005

Hughes:00 Generalising monads to arrows. SCP 37, 2000

Algebraic Approach to computational effects

Plotkin+Power:01 Adequacy for algebraic effects. FOSSACS, 2001

Hyland+Plotkin+Power:02 Combining computational effects. IFIP TCS, 2002

Plotkin+Pretnar:09 Handlers of algebraic effects. ESOP, 2009