TOWARDS A THEORY OF INTENSIONAL SEMANTICS

Stephen Brookes
Shai Geva

Carnegie Mellon University
School of Computer Science
Pittsburgh
Pa 15213

EXTENSIONAL SEMANTICS

EXAMPLES

- state transformation semantics for while-programs
- Scott model of simply typed λ-calculus

FEATURES

- ignores computation details
- models only input-output behavior
- supports reasoning about familiar *extensional* properties
  - Hoare logic for partial correctness
  - LCF
- can use to show correctness-preservation for program transformations
- cannot distinguish between programs with same input-output behavior
- cannot reason about *intensional* properties

INTENSIONAL SEMANTICS

EXAMPLES

- Berry-Curien sequential algorithms on concrete data structures
- Brookes-Geva parallel algorithms on generalized concrete data structures

FEATURES

- models computation strategy
- reasoning about intensional properties
  - order of evaluation
  - degree of parallelism
- can use to show efficiency-improvement of program transformations
- can recover extensional from intensional
- algorithm = function + computation strategy
- intensional models tend to be more concrete

RESEARCH AIMS

Develop a theory of intensional semantics:

- allow semantics at differing levels of abstraction
- show relationships between different models of the same programming language
- show relationship to existing intensional and extensional models
- use intensional semantics to reason about efficiency

THESIS

Category theory provides a general framework:

- extensional semantics in a category $\mathcal{C}$
- datatypes as objects of $\mathcal{C}$
- extensional meanings as morphisms in $\mathcal{C}$
- notion of computation as a comonad T
- intensional semantics in Kleisli category $\mathcal{C}_T$
- intensional meanings as morphisms in $\mathcal{C}_T$

COMONADS

DEFINITION

A *comonad* over a category $\mathcal{C}$ is a (co-)triple (T, ϵ, δ) where

- $T : \mathcal{C} \rightarrow \mathcal{C}$ is a functor
- $\epsilon : T \stackrel{.}{\rightarrow} I_{\mathcal{C}}$ is a natural transformation
- $\delta : T \stackrel{.}{\rightarrow} T^2$ is a natural transformation

and for every object A the following associativity and identity laws hold:

$$\begin{aligned} T(\delta_A) \circ \delta_A &= \delta_{TA} \circ \delta_A \\ \epsilon_{TA} \circ \delta_A &= T(\epsilon_A) \circ \delta_A = \mathsf{id}_{TA}. \end{aligned}$$

INTUITION

- TA is a datatype of *computations over* A.
- ϵt is the value computed by t.
- δt is a computation over TA that computes t.

KLEISLI CATEGORIES

DEFINITION

Given a comonad (T, ϵ, δ) over $\mathcal{C}$, the *Kleisli category* $\mathcal{C}_T$ is defined by:

- The objects of $\mathcal{C}_T$ are the objects of $\mathcal{C}$.
- The morphisms from A to B in $\mathcal{C}_T$ are the morphisms from TA to B in $\mathcal{C}$.
- The identity morphism $\widehat{\mathsf{id}}_A$ on A in $\mathcal{C}_T$ is $\epsilon_A : TA \rightarrow^{\mathcal{C}} A$.
- The composition in $\mathcal{C}_T$ of $a : A \rightarrow^{\mathcal{C}_T} B$ and $a' : B \rightarrow^{\mathcal{C}_T} C$ is

$$a' \bar{\circ}\, a = a' \circ Ta \circ \delta_A.$$

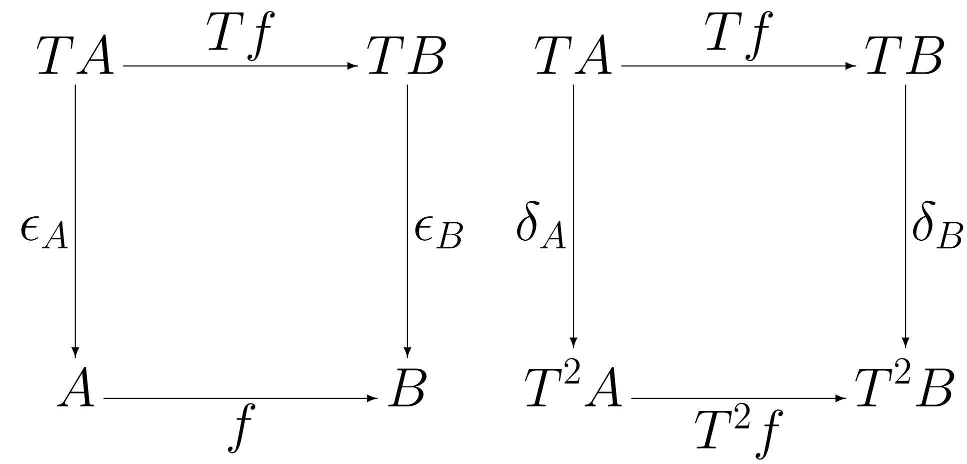
TERMINOLOGY

A morphism in $\mathcal{C}_T$ is an *algorithm*.

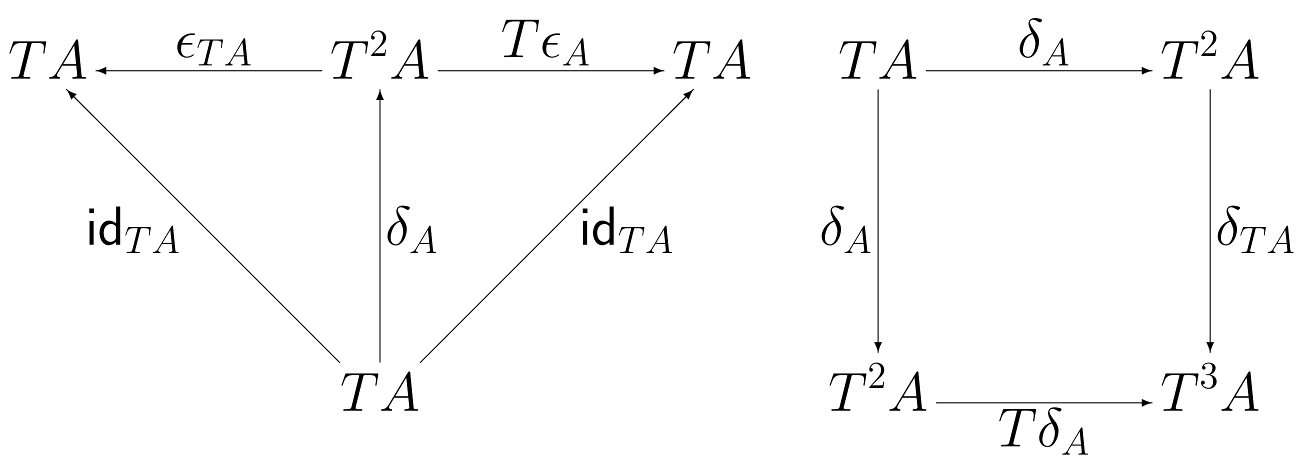
INTUITION

- The intensional meaning of a program is a function from input computations to output values.
- Programs operate in demand-driven, coroutine-like manner (Kahn-MacQueen)
  - program responds to a request for output by performing input computation until it has enough information to determine what output value to produce
- The identity algorithm from A to A simply evaluates its input.
- $a' \bar{\circ} a$ behaves like a' using a to generate input.
- A similar, but sequential, operational model is used by Berry and Curien.

COMONAD DIAGRAMS



Naturality



Identity and associativity laws

COMPUTATIONAL COMONADS

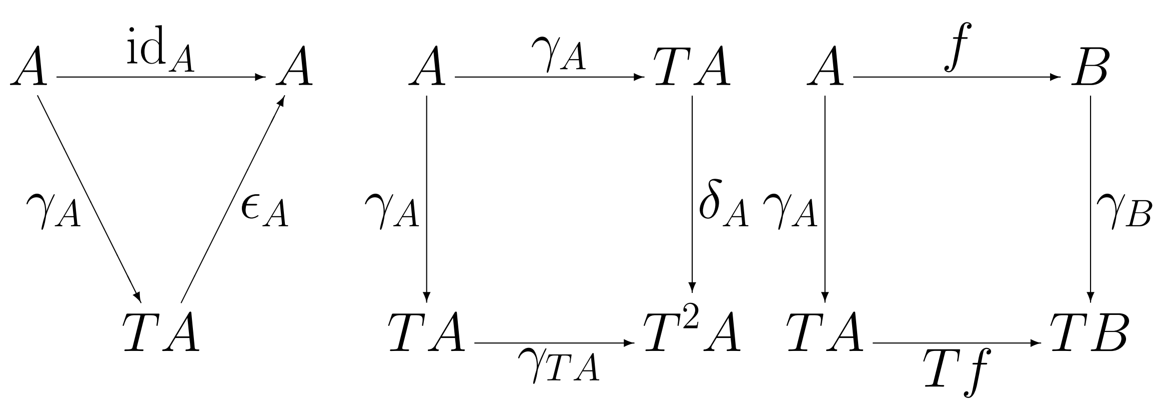
DEFINITION

A *computational comonad* over a category $\mathcal{C}$ is a quadruple $(T, \epsilon, \delta, \gamma)$ such that

- (T, ϵ, δ) is a comonad over $\mathcal{C}$
- $\gamma : I_{\mathcal{C}} \dot{\to} T$ is a natural transformation such that, for every object A,
  - $\epsilon_A \circ \gamma_A = \mathsf{id}_A$
  - $\delta_A \circ \gamma_A = \gamma_{TA} \circ \gamma_A$.

INTUITION

γx is a degenerate computation that computes x.



RELATING ALGORITHMS AND FUNCTIONS

Let $(T, \epsilon, \delta, \gamma)$ be a computational comonad.

DEFINITION

The functors alg and fun between $\mathcal{C}$ and $\mathcal{C}_T$ are given by:

- $\mathsf{alg} : \mathcal{C} \rightarrow \mathcal{C}_T$ is the identity on objects;
- $\mathsf{alg}(f) = f \circ \epsilon_A$, for every $f : A \rightarrow^{\mathcal{C}} B$;
- $\mathsf{fun} : \mathcal{C}_T \rightarrow \mathcal{C}$ is the identity on objects;
- $\mathsf{fun}(a) = a \circ \gamma_A$, for all $a : A \rightarrow^{\mathcal{C}_T} B$.

TERMINOLOGY

- $\mathsf{fun}(a)$ is the input-output function of a.
- $\mathsf{alg}(f)$ is a (canonical) algorithm computing f.

INPUT-OUTPUT EQUIVALENCE

- fun induces an *input-output equivalence* relation $=^{io}$ on $\mathcal{C}_T$:

$$a_1 =^{io} a_2 \iff \mathsf{fun}(a_1) = \mathsf{fun}(a_2).$$

- This relation is a congruence:

$$a_1 =^{io} a_2 \ \&\ a_1' =^{io} a_2' \quad \Rightarrow \quad a_1' \bar{\circ} a_1 =^{io} a_2' \bar{\circ} a_2.$$

- The quotient category of $\mathcal{C}_T$ by $=^{io}$ is isomorphic to $\mathcal{C}$.

- For all $f : A \to^{\mathcal{C}} B$, $\mathsf{fun}(\mathsf{alg}\ f) = f$.

- For all $a : A \to^{\mathcal{C}_T} B$, $\mathsf{alg}(\mathsf{fun}\ a) =^{io} a$.

PROPERTIES of KLEISLI CATEGORIES

PROPOSITION

If $\mathcal{C}$ has distinguished finite products and projections π_i, then $\mathcal{C}_T$ has distinguished finite products, with projections

$$\begin{aligned} \widehat{\pi}_i &: A_1 \times A_2 \rightarrow^{\mathcal{C}_T} A_i \\ \widehat{\pi}_i &= \epsilon_{A_i} \circ T\,\pi_i \\ &= \pi_i \circ \epsilon_{A_1 \times A_2}. \end{aligned}$$

PROPOSITION

If $\mathcal{C}$ is cartesian closed and T preserves finite products, then $\mathcal{C}_T$ is cartesian closed.

[Under more general assumptions the Kleisli category of T is weakly cartesian closed.]

ALGORITHMS ON DOMAINS

- Let **CONT** be the category of Scott domains and continuous functions.
- We define a computational comonad

$$(T, \mathtt{val}, \mathtt{pre}, \mathtt{path})$$

over **CONT**.

- Let **ALG** be the Kleisli category of T.
- Morphisms in **ALG** can be viewed as *parallel algorithms*.
- **CONT** and **ALG** are cartesian closed.
- We give an extensional and an intensional interpretation for simply typed λ-calculus and prove a Correspondence Theorem.
- **CONT** and **ALG** are ordered categories.
- We use $\leq^i$ for the pointwise order on algorithms.
- $\widehat{\mathsf{curry}}$ and $\widehat{\mathsf{app}}$ denote currying and application on algorithms.

COMPUTATION ON DOMAINS

- Let TD be the set of non-decreasing sequences over D, ordered componentwise.
- For $f : D \to D'$, let $Tf : TD \to TD'$ be the function that applies f componentwise:

$$(Tf)\langle d_n \rangle_{n=0}^{\infty} = \langle f d_n \rangle_{n=0}^{\infty}.$$

- For $t \in TD$ let $\mathtt{val}_D\, t$ be the least upper bound of t.
- For $t = \langle d_n \rangle_{n=0}^{\infty}$ let $\mathtt{pre}_D\, t$ be the sequence of prefixes of t: for all $k \geq 0$,

$$(\mathtt{pre}\, t)_k = d_0 \ldots d_k d_k^{\omega}.$$

- For $d \in D$ let $\mathtt{path}\, d = d^{\omega}$.

INTUITION

- A computation in TD records a sequence of incremental steps towards a value in D. Idle steps are permitted.

- Every computation is the limit of its prefixes.

- A degenerate computation consists entirely of idle steps.

PARALLEL-OR

- The most eager algorithm:

$$
\begin{aligned}
\texttt{epPOR}(\langle \bot, T\rangle^{\omega}) &= T \\
\texttt{epPOR}(\langle T, \bot\rangle^{\omega}) &= T \\
\texttt{epPOR}(\langle F, F\rangle^{\omega}) &= F
\end{aligned}
$$

- The laziest algorithm:

$$
\begin{aligned}
\texttt{lpPOR}(\langle \bot, \bot\rangle^{n} \langle \bot, T\rangle^{\omega}) &= T \\
\texttt{lpPOR}(\langle \bot, \bot\rangle^{n} \langle T, \bot\rangle^{\omega}) &= T \\
\texttt{lpPOR}(\langle \bot, \bot\rangle^{n} \langle F, F\rangle^{\omega}) &= F
\end{aligned}
$$

for all $n \geq 0$.

- $\texttt{epPOR} \leq^{i} \texttt{lpPOR}$.

LEFT-STRICT-OR

- The most eager sequential algorithm:

$$\begin{aligned} \texttt{esLOR}(\langle T, \bot\rangle^\omega) &= T \\ \texttt{esLOR}(\langle F, \bot\rangle \langle F, T\rangle^\omega) &= T \\ \texttt{esLOR}(\langle F, \bot\rangle \langle F, F\rangle^\omega) &= F. \end{aligned}$$

- The most eager parallel algorithm:

$$\begin{aligned} \texttt{epLOR}(\langle T, \bot\rangle^\omega) &= T \\ \texttt{epLOR}(\langle F, T\rangle^\omega) &= T \\ \texttt{epLOR}(\langle F, F\rangle^\omega) &= F. \end{aligned}$$

- The laziest parallel algorithm:

$$\begin{aligned} \texttt{lpLOR}(\langle \bot, \bot\rangle^n \langle T, \bot\rangle^\omega) &= T \\ \texttt{lpLOR}(\langle \bot, \bot\rangle^n \langle F, \bot\rangle^m \langle F, T\rangle^\omega) &= T \\ \texttt{lpLOR}(\langle \bot, \bot\rangle^n \langle F, \bot\rangle^m \langle F, F\rangle^\omega) &= F \end{aligned}$$

for all $m, n \geq 0$.

- $\texttt{epLOR} \leq^i \texttt{esLOR} \leq^i \texttt{lpLOR}$.
- $\texttt{epLOR} \leq^i \texttt{epPOR}$.
- $\texttt{lpLOR} \leq^i \texttt{lpPOR}$.

DOUBLY-STRICT-OR

- The most eager parallel algorithm:

$$\begin{aligned}
\texttt{epSOR}(\langle T, T\rangle^{\omega}) &= T \\
\texttt{epSOR}(\langle T, F\rangle^{\omega}) &= T \\
\texttt{epSOR}(\langle F, T\rangle^{\omega}) &= T \\
\texttt{epSOR}(\langle F, F\rangle^{\omega}) &= F.
\end{aligned}$$

- The laziest parallel algorithm:

$$\begin{aligned}
\texttt{lpSOR}(\langle \bot, \bot\rangle^{m} \langle T, \bot\rangle^{n} \langle T, T\rangle^{\omega}) &= T \\
\texttt{lpSOR}(\langle \bot, \bot\rangle^{m} \langle \bot, T\rangle^{n} \langle T, T\rangle^{\omega}) &= T \\
\texttt{lpSOR}(\langle \bot, \bot\rangle^{m} \langle T, \bot\rangle^{n} \langle T, F\rangle^{\omega}) &= T \\
\texttt{lpSOR}(\langle \bot, \bot\rangle^{m} \langle \bot, F\rangle^{n} \langle T, F\rangle^{\omega}) &= T \\
\texttt{lpSOR}(\langle \bot, \bot\rangle^{m} \langle F, \bot\rangle^{n} \langle F, T\rangle^{\omega}) &= T \\
\texttt{lpSOR}(\langle \bot, \bot\rangle^{m} \langle \bot, T\rangle^{n} \langle F, T\rangle^{\omega}) &= T \\
\texttt{lpSOR}(\langle \bot, \bot\rangle^{m} \langle T, \bot\rangle^{n} \langle T, T\rangle^{\omega}) &= T \\
\texttt{lpSOR}(\langle \bot, \bot\rangle^{m} \langle \bot, T\rangle^{n} \langle T, T\rangle^{\omega}) &= T
\end{aligned}$$

for all $m, n \geq 0$.

DOUBLY-STRICT-OR

- The most eager sequential left-right algorithm:

$$
\begin{aligned}
\texttt{elrSOR}(\langle T, \bot\rangle \, \langle T, T\rangle^{\omega}) &= T \\
\texttt{elrSOR}(\langle T, \bot\rangle \, \langle T, F\rangle^{\omega}) &= T \\
\texttt{elrSOR}(\langle F, \bot\rangle \, \langle F, T\rangle^{\omega}) &= T \\
\texttt{elrSOR}(\langle F, \bot\rangle \, \langle F, F\rangle^{\omega}) &= F.
\end{aligned}
$$

- The most eager sequential right-left algorithm:

$$
\begin{aligned}
\texttt{erlSOR}(\langle \bot, T\rangle \, \langle T, T\rangle^{\omega}) &= T \\
\texttt{erlSOR}(\langle \bot, T\rangle \, \langle F, T\rangle^{\omega}) &= T \\
\texttt{erlSOR}(\langle \bot, F\rangle \, \langle T, F\rangle^{\omega}) &= T \\
\texttt{erlSOR}(\langle \bot, F\rangle \, \langle F, F\rangle^{\omega}) &= F.
\end{aligned}
$$

- $\texttt{epSOR} \leq^{i} \texttt{elrSOR} \leq^{i} \texttt{lpSOR}$
- $\texttt{epSOR} \leq^{i} \texttt{erlSOR} \leq^{i} \texttt{lpSOR}$
- `elrSOR` and `erlSOR` are incomparable.

COMPOSITION

$$\begin{aligned}
\texttt{esLOR} &: \mathbf{Bool}^2 \to \mathbf{Bool} \\
\texttt{lpPOR} &: \mathbf{Bool} \times \mathbf{Bool} \to \mathbf{Bool} \\
\widehat{\mathsf{curry}}(\texttt{lpPOR}) &: \mathbf{Bool} \to \mathbf{Bool} \to \mathbf{Bool} \\
\widehat{\mathsf{curry}}(\texttt{lpPOR}) \mathbin{\bar{\circ}} \texttt{esLOR} &: \mathbf{Bool}^2 \to \mathbf{Bool} \to \mathbf{Bool}
\end{aligned}$$

This composite algorithm is characterized by:

$$\begin{aligned}
a(\langle \bot, \bot \rangle^\omega)(\bot^n T^\omega) &= T \\
a(\langle T, \bot \rangle^\omega)(\bot^\omega) &= T \\
a(\langle F, \bot \rangle \, \langle F, T \rangle^\omega)(\bot^\omega) &= T \\
a(\langle F, \bot \rangle \, \langle F, F \rangle^\omega)(\bot^n F^\omega) &= F
\end{aligned}$$

for all $n \geq 0$.

Note the composite computation strategy: eager sequential in the first argument, lazy in the second.

DE MORGAN ALGORITHMS

- Let $\texttt{lNEG}$ and $\texttt{eNEG}$ be the most lazy and most eager algorithms for boolean negation:

$$\begin{aligned}\texttt{lNEG}(\bot^n T^\omega) &= F \quad (n \geq 0)\\ \texttt{lNEG}(\bot^n F^\omega) &= T \quad (n \geq 0)\\ \texttt{eNEG}(T^\omega) = F, &\quad \texttt{eNEG}(F^\omega) = T\end{aligned}$$

- Let $\texttt{dual}$ be the function

$$\lambda a\ .\ \texttt{lNEG} \mathbin{\bar{\circ}} a \mathbin{\bar{\circ}} (\texttt{lNEG} \times \texttt{lNEG}).$$

  This transforms an algorithm a for a binary truth function f into an algorithm for the dual of f, and interchanges the roles of T and F in the computation strategy.

- For example, $\texttt{dual}(\texttt{esLOR}) = \texttt{esLAND}$.

- Let $\texttt{DUAL}$ be the canonical algorithm for $\texttt{dual}$.

- $\texttt{lNEG} \mathbin{\bar{\circ}} \texttt{lNEG}$ is the identity algorithm, and so is $\texttt{DUAL} \mathbin{\bar{\circ}} \texttt{DUAL}$.

- Using $\texttt{eNEG}$ instead of $\texttt{lNEG}$ can alter the computation strategy:

$$\texttt{eNEG} \mathbin{\bar{\circ}} \texttt{lpPOR} \mathbin{\bar{\circ}} (\texttt{eNEG} \times \texttt{eNEG}) = \texttt{epAND}.$$

SIMPLY TYPED LAMBDA CALCULUS

- Let ρ range over a set of *atomic types*.

- The set **Type** of simple *types* is defined by:

$$\tau \ ::= \ \rho \mid \tau_1 \times \tau_2 \mid \tau \to \tau'.$$

- Let c range over a set **Con** of *constants* and X range over a set **Ide** of *identifiers*. Each constant c has a given type τ_c.

- The set of *raw terms* is defined by:

$$\begin{array}{rcl} M & ::= & c \mid X \mid M_1 M_2 \mid \lambda X{:}\tau.M \mid \\ & & (M_1, M_2) \mid \texttt{fst}\ M \mid \texttt{snd}\ M. \end{array}$$

- A *type environment* is a finite partial function w from **Ide** to **Type**.

- A *type judgement* has form $w \vdash M : \tau$.

TYPE JUDGEMENTS

$$\frac{}{w \vdash c : \tau_c}$$

$$\frac{}{w \vdash X : w[\![X]\!]} \quad \text{when } X \in \text{dom}(w)$$

$$\frac{w \vdash M_1 : (\tau \to \tau') \quad w \vdash M_2 : \tau}{w \vdash M_1 M_2 : \tau'}$$

$$\frac{w[\tau/X] \vdash M' : \tau'}{w \vdash (\lambda X{:}\tau.M') : \tau \to \tau'}$$

$$\frac{w \vdash M_1 : \tau_1 \quad w \vdash M_2 : \tau_2}{w \vdash (M_1, M_2) : \tau_1 \times \tau_2}$$

$$\frac{w \vdash M : \tau_1 \times \tau_2}{w \vdash \texttt{fst}\ M : \tau_1} \qquad \frac{w \vdash M : \tau_1 \times \tau_2}{w \vdash \texttt{snd}\ M : \tau_2}$$

TYPE INTERPRETATIONS

- Assume given a domain A_ρ for each atomic type ρ.
- The extensional and intensional type interpretations

$$E = (E_\tau \mid \tau \in \mathbf{Type})$$
$$I = (I_\tau \mid \tau \in \mathbf{Type})$$

  are defined by:

$$\begin{array}{rclcrcl} E_\rho & = & A_\rho & \qquad & I_\rho & = & A_\rho \\ E_{\tau_1 \times \tau_2} & = & E_{\tau_1} \times E_{\tau_2} & & I_{\tau_1 \times \tau_2} & = & I_{\tau_1} \times I_{\tau_2} \\ E_{\tau \to \tau'} & = & E_\tau \to E_{\tau'} & & I_{\tau \to \tau'} & = & I_\tau \to^i I_{\tau'}. \end{array}$$

- Products and exponentiations are taken in **CONT** and **ALG**, respectively.
- $I_\tau \to^i I_{\tau'} = T I_\tau \to I_{\tau'}$.

ENVIRONMENTS

Let w be a type environment.

- An extensional w-environment maps each identifier in scope to an extensional value of appropriate type:

$$\mathrm{Env}_{Ew} = \Pi_{X \in \mathrm{dom}(w)} E_{w[\![X]\!]}.$$

- When $w \vdash M : \tau$ the extensional meaning of M is a function from Env_{Ew} to E_τ.

- An intensional w-environment maps each identifier in scope to an intensional value of appropriate type:

$$\mathrm{Env}_{Iw} = \Pi_{X \in \mathrm{dom}(w)} I_{w[\![X]\!]}.$$

- When $w \vdash M : \tau$ the intensional meaning of M is an algorithm from Env_{Iw} to I_τ.

- Since T preserves finite products,

$$T(\mathrm{Env}_{Iw}) = \Pi_{X \in \mathrm{dom}(w)} T I_{w[\![X]\!]},$$

  so identifiers get bound to *computations* in the intensional semantics.

EXTENSIONAL SEMANTICS

- Assume given an extensional interpretation

  $$[\![c]\!]_E \in E_{\tau_c}$$

  for each constant c.

- The extensional semantics is the family of semantic functions

  $$\mathcal{E}_{w,\tau} : \mathbf{Term}_{w,\tau} \to (\mathrm{Env}_{Ew} \to E_\tau)$$

  defined by:

$$
\begin{array}{rcl}
\mathcal{E}[\![c]\!]\epsilon &=& [\![c]\!]_E \\
\mathcal{E}[\![X]\!]\epsilon &=& \epsilon[\![X]\!] \\
\mathcal{E}[\![M_1 M_2]\!]\epsilon &=& (\mathsf{app} \circ \langle \mathcal{E}[\![M_1]\!], \mathcal{E}[\![M_2]\!] \rangle)\epsilon \\
&=& (\mathcal{E}[\![M_1]\!]\epsilon)(\mathcal{E}[\![M_2]\!]\epsilon) \\
\mathcal{E}[\![\lambda X{:}\tau.M]\!]\epsilon &=& \mathsf{curry}(\mathcal{E}[\![M]\!])\epsilon \\
&=& \lambda e \in E_\tau.\mathcal{E}[\![M]\!](\epsilon[e/X]) \\
\mathcal{E}[\![(M_1, M_2)]\!]\epsilon &=& (\mathcal{E}[\![M_1]\!]\epsilon, \mathcal{E}[\![M_2]\!]\epsilon) \\
\mathcal{E}[\![\mathtt{fst}\ M]\!]\epsilon &=& \pi_1(\mathcal{E}[\![M]\!]\epsilon) \\
\mathcal{E}[\![\mathtt{snd}\ M]\!]\epsilon &=& \pi_2(\mathcal{E}[\![M]\!]\epsilon).
\end{array}
$$

INTENSIONAL SEMANTICS

- Assume given an intensional interpretation

$$[\![c]\!]_I \in I_{\tau_c}$$

for each constant c.

- The intensional semantics is the family of semantic functions

$$\mathcal{I}_{w,\tau} : \mathbf{Term}_{w,\tau} \to (T\mathrm{Env}_{Iw} \to I_\tau)$$

defined by:

$$\begin{aligned}
\mathcal{I}[\![c]\!]\iota &= [\![c]\!]_I \\
\mathcal{I}[\![X]\!]\iota &= \mathtt{val}(\iota[\![X]\!]) \\
\mathcal{I}[\![M_1 M_2]\!]\iota &= (\widehat{\mathsf{app}} \mathbin{\bar\circ} \langle \mathcal{I}[\![M_1]\!], \mathcal{I}[\![M_2]\!]\rangle)\iota \\
&= (\mathcal{I}[\![M_1]\!]\iota)(T(\mathcal{I}[\![M_2]\!])(\mathtt{pre}\,\iota)) \\
\mathcal{I}[\![\lambda X{:}\tau.M]\!]\iota &= \widehat{\mathsf{curry}}(\mathcal{I}[\![M]\!])\iota \\
&= \lambda t \in T I_\tau . \mathcal{I}[\![M]\!](\iota[t/X]) \\
\mathcal{I}[\![(M_1, M_2)]\!]\iota &= (\mathcal{I}[\![M_1]\!]\iota, \mathcal{I}[\![M_2]\!]\iota) \\
\mathcal{I}[\![\mathtt{fst}\ M]\!]\iota &= \pi_1(\mathcal{I}[\![M]\!]\iota) \\
\mathcal{I}[\![\mathtt{snd}\ M]\!]\iota &= \pi_2(\mathcal{I}[\![M]\!]\iota).
\end{aligned}$$

RELATING TYPE INTERPRETATIONS

Define a type-indexed family of relations

$$\sim_\tau \subseteq I_\tau \times E_\tau$$

by:

$$\begin{array}{rcl}
\sim_\rho & = & \mathrm{id}_{A\rho} \\
\sim_{\tau_1 \times \tau_2} & = & \{((i_1, i_2), (e_1, e_2)) \mid i_1 \sim_{\tau_1} e_1 \;\&\; i_2 \sim_{\tau_2} e_2\} \\
\sim_{\tau \to \tau'} & = & \{(a, f) \mid \forall (i, e) \in I_\tau \times E_\tau. \\
& & \qquad i \sim_\tau e \;\Rightarrow\; \mathsf{fun}(a) i \sim_{\tau'} f(e)\}
\end{array}$$

PROPERTIES

- Algorithm compositions relate to function compositions:

  $a \sim_{\tau \to \tau'} f \;\&\; a' \sim_{\tau' \to \tau''} f' \;\Rightarrow\; (a' \bar{\circ} a) \sim_{\tau \to \tau''} (f' \circ f).$

- Currying of algorithms relates to currying of functions:

  $a \sim_{\tau_1 \times \tau_2 \to \tau'} f \;\Rightarrow\; \widehat{\mathsf{curry}}(a) \sim_{\tau_1 \to (\tau_2 \to \tau')} \mathsf{curry}(f).$

RELATING ENVIRONMENTS

Let w be a type environment, $\iota \in T\mathrm{Env}_{Iw}$, $\epsilon \in \mathrm{Env}_{Ew}$.

DEFINITION

$\iota \sim \epsilon$ iff for all $X : \tau \in w$, there is a pair $(i, e) \in I_\tau \times E_\tau$ such that $\iota[\![X]\!] = \mathtt{path}\, i$, $\epsilon[\![X]\!] = e$, and $i \sim_\tau e$.

So ι relates to ϵ iff for all relevant identifiers X, $\iota[\![X]\!]$ is a degenerate computation of an intensional value that relates to the extensional value $\epsilon[\![X]\!]$.

This is similar to a *logical relation*, but not identical because of the use of fun.

RELATING SEMANTICS

INTUITION

Whenever $\iota \sim \epsilon$, the intensional meaning of a well-typed term in ι relates to its extensional meaning in ϵ.

PROPOSITION

- Assume that for each constant c, $[\![c]\!]_I \sim_{\tau_c} [\![c]\!]_E$.
- Then for all $M \in \mathbf{Term}_{w,\tau}$, all $\iota \in T\mathrm{Env}_{Iw}$, and all $\epsilon \in \mathrm{Env}_{Ew}$,

$$\iota \sim \epsilon \;\Rightarrow\; \mathcal{I}[\![M]\!]\iota \sim_\tau \mathcal{E}[\![M]\!]\epsilon.$$

PROOF:
by induction on the proof of $w \vdash M : \tau$.

EXT and INT

DEFINITION

Define two type-indexed families of functions

$$\mathtt{ext}_\tau : I_\tau \longrightarrow E_\tau \qquad \mathtt{int}_\tau : E_\tau \longrightarrow I_\tau$$

by induction on τ:

- For $\rho \in$ **Atomic**, $\mathtt{ext}_\rho$ and $\mathtt{int}_\rho$ are the identity function.
- For product types:

$$\mathtt{ext}_{\tau_1 \times \tau_2} = \mathtt{ext}_{\tau_1} \times \mathtt{ext}_{\tau_2} \qquad \mathtt{int}_{\tau_1 \times \tau_2} = \mathtt{int}_{\tau_1} \times \mathtt{int}_{\tau_2} .$$

- For an exponentiation $\tau \longrightarrow \tau'$ let:

$$\begin{aligned} \mathtt{ext}_{\tau \to \tau'} &= \lambda a \,.\, \mathtt{ext}_{\tau'} \circ \mathsf{fun}(a) \circ \mathtt{int}_\tau \\ \mathtt{int}_{\tau \to \tau'} &= \lambda f \,.\, \mathsf{alg}(\mathtt{int}_{\tau'} \circ f \circ \mathtt{ext}_\tau). \end{aligned}$$

TERMINOLOGY

- $\mathtt{ext}_\tau(a)$ is the *extension* of a.
- $\mathtt{int}_\tau(e)$ is the *intension* of e.

PROPERTIES of EXT and INT

- Atomic types have no (extra) intensional content.

- When τ is a product of atomic types $\mathtt{ext}_{\tau\to\tau'}$ is fun, and $\mathtt{int}_{\tau\to\tau'}$ is alg.

- For each τ, E_τ is a retract of I_τ: for all $e \in E_\tau$ and all $a \in I_\tau$,

$$\begin{aligned} e &= \mathtt{ext}_\tau(\mathtt{int}_\tau\, e), \\ a &\leq^i \mathtt{int}_\tau(\mathtt{ext}_\tau\, a). \end{aligned}$$

Thus every extensional value is the extension of some intensional value.

EXTENSIONAL EQUIVALENCE

DEFINITION

- a_1 is *extensionally below* a_2, written $a_1 \leq^e a_2$, iff $\texttt{ext}_\tau\, a_1 \leq \texttt{ext}_\tau\, a_2$.
- a_1 and a_2 are *extensionally equivalent*, written $a_1 =^e a_2$, iff they have the same extension.

PROPOSITION

- For all $a_1, a_2 \in I_\tau$, $a_1 \leq^i a_2$ implies $a_1 \leq^e a_2$.
- Hence, the quotient of I_τ by extensional equivalence is isomorphic to E_τ, with $\texttt{ext}_\tau$ and $\texttt{int}_\tau$ inducing the isomorphism:

$$(I_\tau, \leq^i)/_{=^e} \;\cong\; (E_\tau, \leq).$$

- For all $a_1, a_2 \in I_{\tau \to \tau'}$, $a_1 \leq^{io} a_2$ implies $a_1 \leq^e a_2$.

CORRESPONDENCE THEOREM

PROPOSITION

For all τ, and all $i \in I_\tau$ and $e \in E_\tau$,

- $i \sim_\tau e \;\Rightarrow\; e = \mathtt{ext}_\tau\, i$.
- $\mathtt{int}_\tau\, e \sim_\tau e$.

COROLLARY

Assume that for all $c \in \mathbf{Con}$, $[\![c]\!]_I \sim_{\tau_c} [\![c]\!]_E$. Then for all $M \in \mathbf{Term}_{w,\tau}$ and all $\iota \in T\mathrm{Env}_{Iw}$, $\epsilon \in \mathrm{Env}_{Ew}$, $\iota \sim \epsilon \;\Rightarrow\; \mathtt{ext}_\tau(\mathcal{I}[\![M]\!]\iota) = \mathcal{E}[\![M]\!]\epsilon$.

INTUITION

- For a well-typed term M and all suitably related intensional and extensional environments, the extensional meaning of M is the extension of its intensional meaning.
- The extensional semantics is faithfully embedded in the intensional semantics.

INTENSIONAL MODELS of PCF

- Choose an intensional interpretation for each PCF constant: e.g.
  - a particular sequential algorithm
  - a most eager parallel algorithm
  - a most lazy parallel algorithm

  for the corresponding function.
- The corresponding intensional model of PCF will relate sensibly to the standard extensional model.
- For any well-typed closed PCF term, the extension of its intensional meaning is the same as its extensional meaning.
- This holds even for terms using the Y-operator, relating the meaning of recursively defined algorithms and functions.

GENERALITY of APPROACH

- Berry-Curien sequential algorithms can be embedded in the parallel algorithms model.

- Can vary the extensional category, e.g.
  - effectively given domains and computable functions
  - concrete domains and sequential functions
  - dI-domains and stable functions

- Can vary the comonad, e.g.
  - non-decreasing sequences
  - increasing sequences
  - finite and infinite sequences
  - timed data

REFERENCES

- *Computational Comonads and Intensional Semantics*, by S. Brookes and S. Geva. In: Applications of Categories in Computer Science, LMS Lecture Notes vol. 177, Cambridge University Press, 1992.

- *Continuous Functions and Parallel Algorithms on Generalized Concrete Data Structures*, by S. Brookes and S. Geva. In: Mathematical Foundations of Programming Semantics (MFPS'91), Springer-Verlag LNCS vol. 598, 1992.

- *A Cartesian Closed Category of Parallel Algorithms on Scott Domains*, by S. Brookes and S. Geva. In: Semantics of Programming Languages and Model Theory, Gordon and Breach Science Publishers, 1992.

- *Towards a Theory of Parallel Algorithms on Concrete Data Structures,* by S. Brookes and S. Geva. In Semantics for Concurrency (Leicester 1990), Springer-Verlag, 1991. Extended version in *Theoretical Computer Science*, 1992.