Specifying syntax with auxiliary operations

Tom Hirschowitz and Ambroise Lafont

October 13, 2022
Motivation

Mathematical operational semantics (Turi and Plotkin, 1997)

Methods in operational semantics → categorical theorems
or
How about doing operational semantics categorically?
Motivation

In Borthelle et al. (2020):

- Abstract congruence result for applicative bisimilarity in higher-order languages.
- Based on Howe’s method.

**Bisimulation candidate**

- Defined by induction on basic $\lambda$-calculus syntax.
- Basic property: closed under substitution.

$\sim$ Need to abstractly specify:

- *basic* operations in the syntax and
- *auxiliary* operations on it.
How to specify both kinds of operations, abstractly?

- However: scope limited to capture-avoiding substitution.
- Goal: cover languages with, e.g., context application (e.g., delimited continuations, higher-order process calculi).

This work

Generalise FPT to auxiliary operations other than substitution. Examples:

- context application: \( E[e] \),
- named substitution in \( \lambda \mu: (e)_{\alpha} f \) (Vaux’s notation),
- partial differentiation in \( \lambda \)-diff: \( \frac{\partial e}{\partial x} \), and so on.
Contributions

- Abstract notion of syntax with auxiliary operations: admissible monad morphism.
- Admissible morphisms from distributive laws.
- Toolkit for generating admissible morphisms:
  - simple and incremental structural laws, and
  - structural equational systems.
Plan

1. Introduction
2. Initial-algebra semantics
3. Admissible monad morphisms
4. Simple structural laws
5. Conclusion
Initial-algebra semantics

- Idea: identify syntactic object **up to isomorphism**, as initial object in a category of models.
- Not tied to any construction method.
- Initiality provides **recursion principle**.
- But needs initial object to exist.
  - Typical case $\Sigma$-alg $\to C$ for endofunctor $\Sigma$ on $C$.
  - More general: $T$-Alg $\to C$ for monad $T$ on $C$.
    Initial algebra: $T\emptyset$.
    (Under suitable hypotheses...)

**For us:**

Syntax, language,... = monad.
Admissible monad morphisms

**Definition**

A monad morphism $\alpha : R \rightarrow S$ is **admissible** iff $\alpha_{\emptyset} : R\emptyset \rightarrow S\emptyset$ is an isomorphism.

**Otherwise said**

$R\emptyset$ is an initial $S$-algebra,

i.e., $S$-operations are admissible in $R\emptyset$. 
Prime class of examples

Fiore, Plotkin, and Turi (FPT), 1999

- Monoidal category $\mathcal{C}$.
- Pointed strong endofunctor $\Sigma$.

$\sim$

- Free monad $(I + \Sigma)^*$ for basic syntax.
- Monad $\Sigma^\otimes$ for whole syntax.

Theorem (FPT, 1999)

The monad morphism $(I + \Sigma)^* \rightarrow \Sigma^\otimes$ is admissible.
Distributive laws

Well-known:

**Proposition**

Any distributive law $\delta : T \circ S \to S \circ T$ induces monad structure on $S \circ T$, with multiplication

$$STST \xrightarrow{S\delta T} SSTT \xrightarrow{\mu^S \mu^T} ST.$$  

Furthermore, $S\eta^T : S \to ST$ is a monad morphism.
Distributive laws

Easy:

Proposition

For any distributive law \( \delta : T \circ S \rightarrow S \circ T \), if \( T\emptyset \cong \emptyset \), then \( S\eta^T : S \rightarrow ST \) is admissible.

Proof.

The morphism \( \eta^T_{\emptyset} : \emptyset \rightarrow T\emptyset \) is an iso.

Intuition for \( T\emptyset \cong \emptyset \): no constants!

Example:

- \( T \) non-empty multiset monad.
- \( T\emptyset \) is non-empty multisets of... nothing.
Admissible morphisms in terms of forgetful functors

- Recall: monadic functor $E \rightarrow C$ means $E \cong T \text{-Alg}$ for some $T$.
- Let Monadic$/C$ denote the full subcategory of CAT$/C$ spanned by monadic functors.
- "Semantics" functor

\[
\text{sem}: \text{Mnd}(C)^{op} \rightarrow \text{Monadic}/C
\]

\[
T \mapsto T \text{-Alg}
\]

\[
(\alpha: S \rightarrow T) \mapsto T \text{-Alg} \rightarrow S \text{-Alg}
\]

\[
(TX \xrightarrow{a} X) \mapsto (SX \xrightarrow{\alpha_X} TX \xrightarrow{a} X).
\]

**Lemma (Barr? Street?)**

The functor $\text{sem}$ is an equivalence of categories.
Admissible morphisms in terms of forgetful functors

Proposition

Given a category $C$ with initial object, monads $S$ and $T$ on $C$, and a monad morphism $\alpha : S \to T$, the following are equivalent:

1. $\alpha$ is admissible;
2. $\text{sem}(\alpha) : T\text{-Alg} \to S\text{-Alg}$ preserves the initial object.
Simple structural laws: Peano

- Consider $\Sigma(X) = 1 + X$, and let $S = \Sigma^*$.
- $S\emptyset = \text{Peano integers}$.
- Auxiliary operation: addition, defined recursively by
  \[
  0 + e = e \quad s(e_1) + e_2 = s(e_1 + e_2).
  \]
  (1)
- Refine arity of addition into bifunctor $\Gamma(X, Y) = X \times Y$ for distinguishing the decreasing occurrence.
- View recursive equations as a natural transformation:

  \[
  \delta_{X,Y} : \Gamma(\Sigma(X), Y) \rightarrow S(\Gamma(X, SY) + X + Y)
  \]

  \[
  0 + y \mapsto y \\
  s(x) + y \mapsto s(x + y)
  \]

  or in full detail:

  \[
  (in_1(\star), y) \mapsto \eta^S(in_3(y)) \\
  (in_2(x), y) \mapsto s(\eta^S(in_1(x, \eta^S(y))))
  \]
Simple structural laws

Let $\Sigma : C \to C$ and $\Gamma : C^2 \to C$, with $S = \Sigma^*$. 

**Definition**

**Simple structural law:** natural transformation

$$\delta_{X,Y} : \Gamma(\Sigma(X), Y) \to S(\Gamma(X, S(Y)) + X + Y).$$
Algebras of a simple structural law

Definition

A \( \delta \)-algebra: object \( X \) with \( a : \Sigma(X) \to X \) and \( b : \Gamma(X, X) \to X \) “satisfying the recursive equation”:

\[
\begin{align*}
\Gamma(\Sigma(X), X) & \xrightarrow{\delta_{X,Y}} \Gamma(a, X) \xrightarrow{\Gamma(a,X)} \Gamma(X, X) \\
S(\Gamma(X, S(X)) + X + X) & \xrightarrow{S(\Gamma(X, a) + X + X)} S(\Gamma(X, X) + X + X) \xrightarrow{S[b, X, X]} S(X) \xrightarrow{\bar{a}} X
\end{align*}
\]

\( \delta \text{-alg} \) = category of \( \delta \)-algebras.

Forgetful functors:

\[
\begin{align*}
\Sigma \text{-alg} & \leftarrow \delta \text{-alg} \\
& \downarrow \quad \downarrow b \\
C & \leftarrow \Sigma \text{-alg}
\end{align*}
\]
Admissible morphisms from simple structural laws

**Theorem**

For any $\delta_{X,Y} : \Gamma(\Sigma(X), Y) \to S(\Gamma(X, S(Y)) + X + Y)$

with $\mathcal{C}$ locally finitely presentable, $\Sigma$ finitary, and $\Gamma$ cocontinuous in its first argument and finitary in its second one,

```
\Sigma \text{-alg} \begin{array}{c}
\text{preserves initial object}
\end{array} \xleftarrow{} \delta \text{-alg}
```

```
\mathcal{C} \quad \text{finitary and monadic}
```

**Corollary**

The induced monad morphism $\Sigma^* \to \delta^*$ is admissible.
Application: sharing $\lambda$-calculus (Accattoli, Kesner,...)

Syntax:

$$e, f ::= x \mid e \ f \mid \lambda x.e \mid e\langle x \mapsto f \rangle$$

$$E ::= \Box \mid E\langle x \mapsto f \rangle$$

Context application:

$$\Box[e] = e$$

$$(E\langle x \mapsto f \rangle)[e] = E[e]\langle x \mapsto f \rangle$$

Warning, variable capture

$$(\Box\langle x \mapsto f \rangle)[x] = x\langle x \mapsto f \rangle.$$
Functor category $[\mathbb{F}, \text{Set}^{1+\mathbb{N}}]$, where $\mathbb{F}$ = finite ordinals and all maps.

**Notation**

- $p = in_1(\star) \in 1 + \mathbb{N}$ (program).
- $c_m = in_2(m) \in 1 + \mathbb{N}$ (context with $m$ capturing variables).
- For any $X \in [\mathbb{F}, \text{Set}^{1+\mathbb{N}}]$, $n \in \mathbb{F}$, $m \in \mathbb{N}$:
  - $X(n)_p$: programs with $n$ free variables, and
  - $X(n)_{c_m}$: contexts with $n$ free variables and $m$ capturing variables.
Syntax (with notation)

Programs

\[
\Sigma(X)(n)_p = n + X(n)^2_p + X(n+1)_p + X(n+1)_p \times X(n)_p \\
\]

\[
e ::= x \mid e \ e \mid \lambda(e) \mid e(e)
\]

Contexts

\[
\Sigma(X)(n)_{c_{1+m}} = X(n+1)_{c_m} \times X(n)_p \\
\Sigma(X)(n)_{c_0} = 1
\]

\[
E ::= E\langle x \mapsto e \rangle \\
\]
Syntax (with notation)

Contexts
\[ \Sigma(X)(n)_{c_{1+m}} = X(n + 1)_{c_m} \times X(n)_p \]
\[ \Sigma(X)(n)_{c_0} = 1 \]

Remark: typing \( E\langle x \mapsto e \rangle \).

- \( \Sigma \)-algebra structure on \( X \) yields
  \[ X(n + 1)_{c_m} \times X(n)_p \rightarrow X(n)_{c_{1+m}} \]
  \[ (E, f) \quad \mapsto \quad E\langle x \mapsto f \rangle \]
- \( x \) is free in \( E \), but capturing in \( E\langle x \mapsto f \rangle \).
- Think of
  \[
  \begin{aligned}
  n &; m \quad \text{as} \quad 1, \ldots, n &; \quad n + 1, \ldots, n + m \\
  n + 1 &; m \quad \text{as} \quad 1, \ldots, n + 1 &; \quad n + 1 + 1, \ldots, n + 1 + m \\
  n &; m + 1 \quad \text{as} \quad 1, \ldots, n &; \quad n + 1, n + 2, \ldots, n + (m + 1)
  \end{aligned}
  \]
Simple structural law

Take
\[ \Gamma(X, Y)(n)_{c_m} = \emptyset \]
and
\[ \Gamma(X, Y)(n)_p = \sum_{m \in \mathbb{N}} X(n)_{c_m} \times Y(n + m)_p \]

\[ \Sigma(X)(n)_{c_m} \times Y(n + m)_p \rightarrow S(\Gamma(X, S(Y)) + X + Y)(n)_p \]

\[ \Box, e \mapsto e \quad (\text{if } m = 0) \]
\[ (E\langle f \rangle, e) \mapsto E[e] \langle f \rangle \quad (\text{if } m = m' + 1), \]

for all \( n, m \in \mathbb{N} \).

Remark

Both \( E \begin{bmatrix} e \end{bmatrix} \) and \( E\langle f \rangle \begin{bmatrix} e \end{bmatrix} \) are valid.

(\( n+1, m' \) \( n+1+m' \) \( n, 1+m' \) \( n+1+m' \))
Characterisation of induced monad

Let $\delta_{X,Y} : \Gamma(\Sigma(X), Y) \to S(\Gamma(X, S(Y)) + X + Y)$ as in the theorem.

**Theorem**

The monad corresponding to $\delta_{\text{alg}} \to C$ is induced by a distributive law

$$TS \to ST,$$

with $T = (X \mapsto \Gamma(X, SX))^*$

Intuition: auxiliary operations, with arbitrary $S$-terms in non-decreasing occurrences.
Conclusion

In this talk:

- **Admissible monad morphisms.**
- Distributive laws $\rightarrow$ admissible monad morphisms.
- Simple structural laws $\rightarrow$ admissible monad morphisms.
- Application: context application in sharing $\lambda$-calculus.

Not in this talk:

- Incremental structural laws $\rightarrow$ admissible monad morphisms.
- Structural equational systems: **benign**, admissible quotienting.
- Applications:
  - named substitution in $\lambda\mu$,
  - partial differentiation in $\lambda$-diff,
  - Fiore, Plotkin, and Turi.

Nowhere yet: quotienting basic syntax, compatibly with auxiliary operations.
Indices in $E\langle f \rangle$

- Let $E_1 = (x_1, x_2; x_3 \vdash \Box(x_3 \mapsto f) : c)$.
- Let $E = E_1 \langle x_2 \mapsto g \rangle = (x_1; x_2, x_3 \vdash \Box(x_3 \mapsto f) \langle x_2 \mapsto g \rangle : c)$.
- Let $e = (x_1, x_2, x_3 \vdash x_1 \ x_2 \ x_3 : p)$.

Form

- $x_1, x_2 \vdash E_1[e] = (x_1 \ x_2 \ x_3) \langle x_3 \mapsto f \rangle : p$
- $x_1 \vdash E[e] = (x_1 \ x_2 \ x_3) \langle x_3 \mapsto f \rangle \langle x_2 \mapsto g \rangle : p$. 