

# Grounding Game Semantics in Categorical Algebra

Jérémie Koenig

Yale University

July 14, 2021

- 1 Context: Building reliable computer systems
- 2 Background: Game semantics and algebraic effects
- 3 Result: Strategies for algebraic effects
- 4 Conclusion: Towards algebraic game semantics

# Section 1

Context: Building reliable computer systems

# Making computers reliable is hard

In modern computer systems:

- Layers upon layers of complex hardware and software components
- A bug in any one of them could prevent the system from working
- Components may break in subtle ways through their interactions

# Making computers reliable is hard

In modern computer systems:

- Layers upon layers of complex hardware and software components
- A bug in any one of them could prevent the system from working
- Components may break in subtle ways through their interactions

Thankfully, there are ways to control this:

- Precise specifications for each component
- Careful and systematic testing
- Formal verification

# Formal verification of software components

To prove a program correct:

- Start with a model of the programming language
- Make the specification mathematically precise
- Write a proof showing that the program indeed meets the specification

# Formal verification of software components

To prove a program correct:

- Start with a model of the programming language
- Make the specification mathematically precise
- Write a proof showing that the program indeed meets the specification

Mechanizing this process in a proof assistant has many advantages:

- Almost no possibility of mistake in the proof
- Can scale up the methodology to complex programs
- The proof can easily be checked by a third-party (*certified* software)

# State of the art and next steps

Over the past  $\sim 10$  years, verification has become increasingly tractable:

- Researchers have verified complex components of various kinds
- Industrial-strength verification tools exist

# State of the art and next steps

Over the past  $\sim 10$  years, verification has become increasingly tractable:

- Researchers have verified complex components of various kinds
- Industrial-strength verification tools exist

The next step is *end-to-end* verification:

- Until then, bugs can sneak into the “gaps” between correctness proofs
- Solution: use formal specifications as *interfaces* to connect proofs
- Challenge: existing projects use different models and proof methods
- This is by *necessity* and not by accident

# End-to-end verification using a hierarchy of models

This suggests we should organize semantic models into a *hierarchy*:

- Individual components are verified using specialized models
- Embed these models into increasingly general ones where certified component can be assembled into certified systems

# End-to-end verification using a hierarchy of models

This suggests we should organize semantic models into a *hierarchy*:

- Individual components are verified using specialized models
- Embed these models into increasingly general ones where certified component can be assembled into certified systems

Category theory provides a unified framework to:

- Characterize existing models
- Establish connections between them
- Guide the design of more general ones

# This paper

I will present a very small step in this direction, looking at connections between two important lines of work:

- *Game semantics* expresses the behavior of program components as *strategies* in games derived from their types;
- *Algebraic effects* model computations with side-effects as *terms* in an algebraic theory.

# This paper

I will present a very small step in this direction, looking at connections between two important lines of work:

- *Game semantics* expresses the behavior of program components as *strategies* in games derived from their types;
- *Algebraic effects* model computations with side-effects as *terms* in an algebraic theory.

I will show that:

- Simple strategies can be used to construct interesting models of algebraic effects
- Conversely, we can take inspiration from algebraic effects to characterize these simple strategies categorically.

# This paper

I will present a very small step in this direction, looking at connections between two important lines of work:

- *Game semantics* expresses the behavior of program components as *strategies* in games derived from their types;
- *Algebraic effects* model computations with side-effects as *terms* in an algebraic theory.

I will show that:

- Simple strategies can be used to construct interesting models of algebraic effects
- Conversely, we can take inspiration from algebraic effects to characterize these simple strategies categorically.

I hope this correspondence can be extended in the future to formulate a more general algebraic approach to game semantics.

## Section 2

# Background: Game semantics and algebraic effects

# Game Semantics

*Game semantics* is a general approach to programming language semantics:

- Types are two-player *games* between a component and its environment.
- Programs of a given type are *strategies* for the corresponding game.

# Game Semantics

*Game semantics* is a general approach to programming language semantics:

- Types are two-player *games* between a component and its environment.
- Programs of a given type are *strategies* for the corresponding game.

This approach is very compelling for heterogeneous verification:

- Games provide a very general notion of interface
- “Rely-guarantee” flavor facilitates compositional reasoning

# Game Semantics

*Game semantics* is a general approach to programming language semantics:

- Types are two-player *games* between a component and its environment.
- Programs of a given type are *strategies* for the corresponding game.

This approach is very compelling for heterogeneous verification:

- Games provide a very general notion of interface
- “Rely-guarantee” flavor facilitates compositional reasoning

However there are challenges to overcome:

- Huge variety of constructions for games and strategies
- Often too complex to formalize in a proof assistant
- Existing work rarely focuses on specifications and verification

# Algebraic Effects

*Algebraic effects* address the narrower problem of *computational side-effects*:

- The available side-effects are given by an algebraic theory
- *Terms* in the theory represent computations, which proceed inwards.
- *Operations* represent effects, their arguments are the possible continuations.

# Algebraic Effects

*Algebraic effects* address the narrower problem of *computational side-effects*:

- The available side-effects are given by an algebraic theory
- *Terms* in the theory represent computations, which proceed inwards.
- *Operations* represent effects, their arguments are the possible continuations.

Advantages:

- Composing effect theories is easier than in the monadic approach
- The framework is simple and systematic, grounded in categorical algebra

Limitations:

- Narrower scope than game semantics, less generality

# Algebraic signatures can be read as games

In the algebraic framework, a program with side-effects:

```
greeting(*) := (if readbit then print "Hi" else print "Hello"); stop
```

is modeled in the following way:

$$\Sigma := \{ \text{readbit} : 2, \text{print}[s] : 1, \text{done} : 0 \mid s \in \text{string} \}$$
$$t := \text{readbit}(\text{print}["\text{Hi}"](\text{stop}), \text{print}["\text{Hello}"](\text{stop}))$$

# Algebraic signatures can be read as games

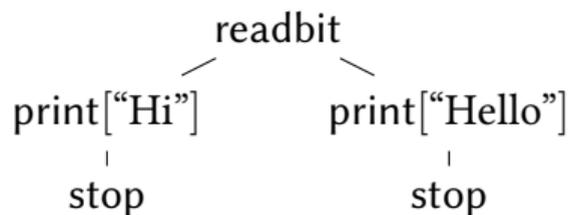
In the algebraic framework, a program with side-effects:

$$\text{greeting}(*) := (\text{if readbit then print "Hi" else print "Hello"}); \text{stop}$$

is modeled in the following way:

$$\begin{aligned} \Sigma &:= \{ \text{readbit} : 2, \text{print}[s] : 1, \text{done} : 0 \mid s \in \text{string} \} \\ t &:= \text{readbit}(\text{print}["\text{Hi}"](\text{stop}), \text{print}["\text{Hello}"](\text{stop})) \end{aligned}$$

Reading the signature  $\Sigma$  as a game, the term  $t$  becomes a *strategy tree*:



# Effect signatures

## Definition (Effect signature)

An *effect signature* is a set  $E$  of operations together with a map  $\text{ar} : E \rightarrow \mathbf{Set}$  which assigns an *arity set* to each one.

# Effect signatures

## Definition (Effect signature)

An *effect signature* is a set  $E$  of operations together with a map  $\text{ar} : E \rightarrow \mathbf{Set}$  which assigns an *arity set* to each one.

## Definition (Algebraic interpretation)

The *terms* in  $E$  with variables in the set  $X$  are defined by the grammar:

$$t \in E^* X ::= \underline{x} \mid \underline{m}(t_n)_{n \in \text{ar}(m)} \quad (m \in E, x \in X)$$

# Effect signatures

## Definition (Effect signature)

An *effect signature* is a set  $E$  of operations together with a map  $\text{ar} : E \rightarrow \mathbf{Set}$  which assigns an *arity set* to each one.

## Definition (Algebraic interpretation)

The *terms* in  $E$  with variables in the set  $X$  are defined by the grammar:

$$t \in E^* X ::= \underline{x} \mid \underline{m}(t_n)_{n \in \text{ar}(m)} \quad (m \in E, x \in X)$$

## Definition (Game interpretation)

The *plays* over an effect signature  $E$  with results in  $X$  are defined by the grammar:

$$s \in P_E(X) ::= \underline{x} \mid \underline{m} \mid \underline{mns} \quad (x \in X, m \in E, n \in \text{ar}(m))$$

# Categorical characterization of terms and strategies

An effect signature can be interpreted as a polynomial endofunctor  $E : \mathbf{Set} \rightarrow \mathbf{Set}$  constructing the terms of depth one:

$$EX := \sum_{m \in E} \prod_{n \in \text{ar}(m)} X$$

An *algebra* for  $E$  is a set  $A$  with a function  $\alpha : EA \rightarrow A$ ; they form a category  $\mathbf{Set}^E$ .

# Categorical characterization of terms and strategies

An effect signature can be interpreted as a polynomial endofunctor  $E : \mathbf{Set} \rightarrow \mathbf{Set}$  constructing the terms of depth one:

$$EX := \sum_{m \in E} \prod_{n \in \text{ar}(m)} X$$

An *algebra* for  $E$  is a set  $A$  with a function  $\alpha : EA \rightarrow A$ ; they form a category  $\mathbf{Set}^E$ .

As is well-known, the “carrier set” functor  $U : \mathbf{Set}^E \rightarrow \mathbf{Set}$  has a left adjoint, which maps a set  $X$  to the term algebra with carrier set  $E^*X$ .

# Categorical characterization of terms and strategies

An effect signature can be interpreted as a polynomial endofunctor  $E : \mathbf{Set} \rightarrow \mathbf{Set}$  constructing the terms of depth one:

$$EX := \sum_{m \in E} \prod_{n \in \text{ar}(m)} X$$

An *algebra* for  $E$  is a set  $A$  with a function  $\alpha : EA \rightarrow A$ ; they form a category  $\mathbf{Set}^E$ .

As is well-known, the “carrier set” functor  $U : \mathbf{Set}^E \rightarrow \mathbf{Set}$  has a left adjoint, which maps a set  $X$  to the term algebra with carrier set  $E^*X$ .

By working in a category of directed-complete partial orders, I obtain a similar characterization for the *strategies* over  $E$ .

## Section 3

Result: Strategies for algebraic effects

# Strategies over an effect signature

# Strategies over an effect signature

## Definition (Coherent plays)

The *coherence* relation  $\circ \subseteq P_E(X) \times P_E(X)$  is the smallest relation satisfying:

$$\begin{array}{l} \underline{x} \circ \underline{x} \qquad \underline{m} \circ \underline{m} \qquad \underline{m} \circ \underline{mns} \\ (n_1 = n_2 \Rightarrow s_1 \circ s_2) \Rightarrow \underline{m} n_1 s_1 \circ \underline{m} n_2 s_2 \end{array}$$

# Strategies over an effect signature

## Definition (Coherent plays)

The *coherence* relation  $\circ \subseteq P_E(X) \times P_E(X)$  is the smallest relation satisfying:

$$\begin{aligned} \underline{x} \circ \underline{x} \qquad \underline{m} \circ \underline{m} \qquad \underline{m} \circ \underline{mns} \\ (n_1 = n_2 \Rightarrow s_1 \circ s_2) \Rightarrow \underline{m} n_1 s_1 \circ \underline{m} n_2 s_2 \end{aligned}$$

## Definition (Effect strategy)

A *strategy*  $\sigma \in S_E(X)$  over a signature  $E$  with results in  $X$  is a prefix-closed set  $\sigma \subseteq P_E(X)$  of pairwise coherent plays.

# Algebraic characterization of strategies

Strategies under set inclusion form a pointed directed-complete partial order:

- The empty strategy is the least element
- Unions of directed sets of strategies preserve the coherence condition

# Algebraic characterization of strategies

Strategies under set inclusion form a pointed directed-complete partial order:

- The empty strategy is the least element
- Unions of directed sets of strategies preserve the coherence condition

It turns out the strategies for  $E$  can be characterized as free algebras in  $\mathbf{DCPO}_{\perp!}$ , where the effect signature  $E$  is interpreted as the endofunctor:

$$\hat{E}X := \bigoplus_{m \in E} \left( \prod_{n \in \text{ar}(m)} X \right)_{\perp}$$

# Algebraic characterization of strategies

Strategies under set inclusion form a pointed directed-complete partial order:

- The empty strategy is the least element
- Unions of directed sets of strategies preserve the coherence condition

It turns out the strategies for  $E$  can be characterized as free algebras in  $\mathbf{DCPO}_{\perp!}$ , where the effect signature  $E$  is interpreted as the endofunctor:

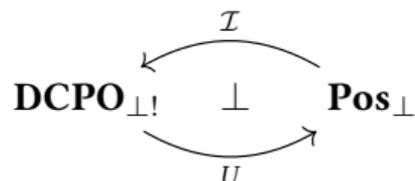
$$\hat{E}X := \bigoplus_{m \in E} \left( \prod_{n \in \text{ar}(m)} X \right)_{\perp}$$

## Theorem

*The forgetful functor  $\hat{U} : \mathbf{DCPO}_{\perp!}^{\hat{E}} \rightarrow \mathbf{Set}$  has a left adjoint.  
The pointed dcpo  $S_E(X)$  carries the corresponding  $\hat{E}$ -algebra.*

# Strategies are ideal completions of terms

The *ideal completion*  $\mathcal{I}$  constructs the free dcpo on a poset:



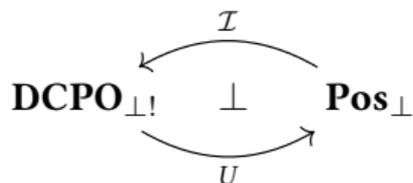
If we order terms with variables in  $X_{\perp}$  using the rules

$$\frac{}{\underline{\perp} \sqsubseteq t} \quad \frac{}{\underline{x} \sqsubseteq x} \quad \frac{\forall n \in \text{ar}(m) \cdot t_n \sqsubseteq t'_n}{\underline{m}(t_n)_{n \in \text{ar}(m)} \sqsubseteq \underline{m}(t'_n)_{n \in \text{ar}(m)}} ,$$

this provides an alternative construction of strategies as  $\mathcal{IE}^*(X_{\perp})$ .

# Strategies are ideal completions of terms

The *ideal completion*  $\mathcal{I}$  constructs the free dcpo on a poset:



If we order terms with variables in  $X_{\perp}$  using the rules

$$\frac{}{\underline{\perp} \sqsubseteq t} \quad \frac{}{\underline{x} \sqsubseteq x} \quad \frac{\forall n \in \text{ar}(m) \cdot t_n \sqsubseteq t'_n}{\underline{m}(t_n)_{n \in \text{ar}(m)} \sqsubseteq \underline{m}(t'_n)_{n \in \text{ar}(m)}} ,$$

this provides an alternative construction of strategies as  $\mathcal{I}E^*(X_{\perp})$ .

## Theorem

The following partial orders are isomorphic:  $S_E(X) \cong \mathcal{I}E^*(X_{\perp}) \cong \mu Y \cdot \hat{E}Y \oplus X_{\perp}$

## Section 4

Conclusion: Towards algebraic game semantics

# Conclusion

Algebraic effects and game semantics have themes in common, but they look at them very differently.

# Conclusion

Algebraic effects and game semantics have themes in common, but they look at them very differently.

I believe we can build on the correspondence I have presented to confront these point of views in interesting ways.

# Conclusion

Algebraic effects and game semantics have themes in common, but they look at them very differently.

I believe we can build on the correspondence I have presented to confront these point of views in interesting ways.

For example:

- Generalizing to multi-sorted signatures allows for richer games. This could serve as a common low-level algebraic grounding for a variety of sequential game semantics models.

# Conclusion

Algebraic effects and game semantics have themes in common, but they look at them very differently.

I believe we can build on the correspondence I have presented to confront these point of views in interesting ways.

For example:

- Generalizing to multi-sorted signatures allows for richer games. This could serve as a common low-level algebraic grounding for a variety of sequential game semantics models.
- Effect signatures and natural transformations  $\eta_X : EX \rightarrow FX \in \mathbf{Set}$  form a symmetric monoidal closed category. Endofunctor composition and the free monad construction can be defined directly on signatures. We can carry out a version of Reddy's *object-based semantics* in this setting.