Grounding Game Semantics in Categorical Algebra

Jérémie Koenig
Yale University, USA
jeremie.koenig@yale.edu

I present a formal connection between *algebraic effects* and *game semantics*, two important lines of work in programming languages semantics with applications in compositional software verification. Specifically, the algebraic signature enumerating the possible side-effects of a computation can be read as a game; strategies for this game constitute the free algebra for the effect signature in a category of complete partial orders (*cpos*).

Hence, strategies provide a convenient model of computations with uninterpreted side-effects. In particular, the operational flavor of game semantics carries over to the algebraic context, in the form of the coincidence between the initial algebras and the terminal coalgebras of cpo endofunctors.

Conversely, the algebraic point of view sheds new light on the strategy constructions underlying game semantics. Strategy models can be reformulated as ideal completions of partial strategy trees (free dcpos on the term algebra). Extending the framework to multi-sorted signatures makes this construction available for a large class of games.

1 Introduction

Writing bug-free software is notoriously hard. Current practice encourages comprehensive testing, but while testing can reveal bugs it can never completely guarantee their absence. Therefore, for critical systems, *verification* has become the gold standard: the desired behavior is described as a mathematical specification, against which the implemented system is formally proven correct [40].

Over the past decade, researchers have been able to apply this methodology to larger and larger systems: there are now verified compilers [32, 43, 30], operating system kernels [26, 21, 22], and even verified processor designs [11, 17]. As a result, the construction of large-scale, heterogeneous computer systems which are fully verified is now within reach [12]. A system of this kind would be described end-to-end by a mathematical model, and certified correct by a computer-checked proof, providing a strong guarantee that a given combination of hardware and software components behaves as expected.

Unfortunately, composing certified components into certified systems is difficult. For verification to be tractable, the models and techniques used must often be tailored to the component at hand. As a result, given two certified components developed independently, it is often challenging to interface their proofs of correctness to construct a larger proof encompassing them both. To facilitate this process, a key task will be to establish a *hierarchy* of common models. Using this hierarchy, individual certified components could continue to use specialized models, but these models could then be embedded into more general ones, where components and proofs of different kinds would be made interoperable.

Category theory is an important tool for this task. It can help us characterize existing models and compare them in a common framework. As a systematic study of compositional structures, it can then guide the design of more general models capable of describing heterogeneous systems. This paper proposes to use this methodology to synthesize two related but distinct lines of work:

- *Algebraic effects* [38, 39] offer a computational reading of basic concepts in categorical algebra for the purpose of modeling, combining, and reasoning about side-effects in computations. They

Submitted to:
ACT 2021

are a principled solution grounded in well-established mathematics, and have prompted novel and promising approaches to programming language design.

- *Game semantics* [2, 7] describes the interface of a program component as a *game* played between the component and its environment, characterizing the component's behavior as a *strategy* in this game. This approach has made it possible to give compositional semantics to existing language features which had previously resisted a satisfactory treatment.

The theory of algebraic effects is outlined in section 2. After introducing game semantics, section 3 uses the associated techniques to construct a *strategy* model of uninterpreted algebraic effects. This model can be characterized as an initial algebra in a particular category of complete partial orders, and reformulated as a completion of the term algebra. Section 4 proposes extending this construction to a large class of games by considering multi-sorted effect signatures.

I will use the notations $\mathbb{1} := \{*\}$ and $\mathbb{2} := \{\mathsf{tt}, \mathsf{ff}\}$. The set of finite sequences over an alphabet Σ is written Σ^*, with ε as the empty sequence and $s \cdot t$ as the concatenation of the sequences s and t. Since the mathematics presented here are ultimately intended to be mechanized in a proof assistant, I will often prefer the use of inductive grammars to that of sets of sequences.

2 Models of computational side-effects

Modeling the *side-effects* of computer programs is a long-standing research topic in programming language semantics. I begin this paper by summarizing the underlying issues, and the recently introduced theory of *algebraic effects* [38] which has led to important developments.

2.1 Monadic effects

Programs which perform pure calculations are straightforward to interpret mathematically. For example,

$$\mathrm{abs}(x) := \textbf{if } x > 0 \textbf{ then return } x \textbf{ else return } -x \tag{1}$$

can be characterized using the function $f : \mathbb{R} \to \mathbb{R}$ which maps x to $|x|$. By contrast, consider the program

$$\mathsf{greeting}(*) := (\textbf{if } \mathsf{readbit} \textbf{ then } \mathsf{print}\ \text{“Hi”} \textbf{ else } \mathsf{print}\ \text{“Hello”})\,;\mathsf{stop} \tag{2}$$

which reads a single bit of input, outputs “Hi” or “Hello” depending on the value of that bit, then terminates without producing a value. The *side-effects* performed by the operations $\mathsf{readbit}$, print and stop are more difficult to model. Certainly, (2) cannot be described as a function $g : \mathbb{1} \to \varnothing$.

The traditional way to address this issue is to capture the available side-effects in a *monad* $\langle T, \eta, \mu \rangle$ [36]. Then TX is used to represent computations with a result in X, but which may also perform side-effects. The monad's unit $\eta : X \to TX$ corresponds to **return**, a pure computation which terminates immediately. The multiplication $\mu : TTX \to TX$ first performs the effects of the outer computation, then those of the computation it evaluates to. This allows us to compose the computations $f : A \to TB$ and $g : B \to TC$ sequentially (;) by using their Kleisli composition $\mu \circ Tg \circ f$.

Example 1. *To assign a meaning to the program (2), we can use the following monad in* **Set***:*

$$TX := (\Sigma^* \times X_\perp)^{\mathbb{2}} \qquad \eta(x) := b \mapsto (\varepsilon, x) \qquad \mu\big(i \mapsto \big(s_i, j \mapsto (s'_{ij}, x_{ij})\big)\big) := b \mapsto \big(s_b \cdot s'_{bb},\, x_{bb}\big)$$

An element of TX is a function which takes as input the bit to be read by readbit. *In addition to the computation's result, which can be $\bot$ as well as a result in X, the function produces a sequence of characters. The operations* readbit, print *and* stop *can be interpreted as:*

$$\begin{array}{ccc} \mathsf{readbit} \in T\mathbb{2} & \mathsf{print} : \Sigma^* \to T\mathbb{1} & \mathsf{stop} \in T\varnothing \\ \mathsf{readbit} := b \mapsto (\varepsilon, b) & \mathsf{print}(s) := b \mapsto (s, *) & \mathsf{stop} := b \mapsto (\varepsilon, \bot) \end{array}$$

Then, the program (2) can be characterized using the function $g : \mathbb{1} \to T\varnothing$ defined by:

$$g(*) := b \mapsto \begin{cases} (\text{“Hi”}, \bot) & \text{if } b = \mathsf{tt} \\ (\text{“Hello”}, \bot) & \text{if } b = \mathsf{ff} \end{cases}$$

A long-standing issue with this approach is that in general monads do not compose. This makes it difficult to combine programs which use different kinds of side-effects.

2.2 Algebraic effects

This can be addressed by restricting our attention to monads describing *algebraic* effects. Computations with side-effects are then seen as *terms* in an algebra. Function symbols correspond to the available effects. Their arities correspond to the number of possible outcomes of the effect, and each argument specifies how the computation will continue should the corresponding outcome occur.

Example 2. *To interpret our running example, the algebraic signature must contain the function symbols* $\mathsf{readbit} : 2$*,* $\mathsf{print}[s] : 1$ *and* $\mathsf{stop} : 0$*. The behavior of the program (2) can then be represented as the term:*

$$\mathsf{readbit}\big(\mathsf{print}[\text{“Hi”}](\mathsf{stop}),\, \mathsf{print}[\text{“Hello”}](\mathsf{stop})\big)$$

and visualized as the tree:

$$\begin{tikzcd} & \mathsf{readbit} \arrow[dl, dash] \arrow[dr, dash] & \\ \mathsf{print}[\text{“Hi”}] \arrow[d, dash] & & \mathsf{print}[\text{“Hello”}] \arrow[d, dash] \\ \mathsf{stop} & & \mathsf{stop} \end{tikzcd}$$

Note that $\mathsf{print}[s]$ *corresponds to a* family *of operations indexed by a parameter $s \in \Sigma^*$.*

A major advantage of this approach is that the basic framework of universal algebra can immediately be brought to bear. For example, equational theories including statements such as:

$$\mathsf{print}[s](\mathsf{print}[s'](x)) = \mathsf{print}[s \cdot s'](x)$$

can be used to characterize the behavior of the different effects and their possible interactions. Algebraic theories can be combined in various ways [25], making possible a compositional treatment of effects. Below, I present a simple version of the approach, I start with the following notion of effect signature.

Definition 3. *An* effect signature *is a set E of function symbols together with a mapping* $\mathsf{ar} : E \to \mathbf{Set}$ *which assigns to each function symbol $m \in E$ an* arity set $\mathsf{ar}(m)$. *I will use the notation*

$$E = \{m_1 : N_1,\, m_2 : N_2,\, \ldots\}$$

where $N_i = \mathsf{ar}(m_i)$ is the arity set associated to the function symbol m_i.

The use of arity *sets* allow us to encode effects such as $\mathsf{readnat} : \mathbb{N}$ which have an infinite number of possible outcomes. In this case the argument tuples will be families indexed by $\mathbb{N}$ and the corresponding terms will be written as $\mathsf{readnat}(x_n)_{n \in \mathbb{N}}$.

2.3 Initial algebras

To give a categorical account of the algebras generated by an effect signature E, we start by interpreting the signature as an endofunctor on **Set**.

Definition 4. *The endofunctor* $\tilde{E} : \mathbf{Set} \to \mathbf{Set}$ *associated with an effect signature* E *is defined as:*

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

When there is no ambiguity we will write both the signature and the endofunctor as E.

The elements of EX are terms of depth one with variables in X. This is emphasized by the notation:

$$t \in EX ::= \underline{m}(x_n)_{n \in \mathsf{ar}(m)} \qquad (m \in E, x \in X)$$

Terms of a fixed depth k can be obtained by iterating the endofunctor as $E^k X$. More generally, the set of all finite terms over the signature can be defined as follows.

Definition 5. *Finite terms over an effect signature* E *with variables in* X *are generated by the grammar:*

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

Interpretations of the signature E in a carrier set A are algebras $\alpha : EA \to A$ for the endofunctor $\tilde{E}$. They can be decomposed into the cotuple $\alpha = [\alpha^m]_{m \in E}$ where $\alpha^m = \alpha \circ \iota_m : A^{\mathsf{ar}(m)} \to A$. They form a category $\mathbf{Set}^E$ where the morphisms of type $\langle A, \alpha \rangle \to \langle B, \beta \rangle$ are the functions $f : A \to B$ satisfying:

$$\begin{tikzcd} EA \arrow[r, "\alpha"] \arrow[d, "Ef"'] & A \arrow[d, "f"] \\ EB \arrow[r, "\beta"] & B \end{tikzcd} \qquad f \circ \alpha = \beta \circ Ef$$

It is well-known [41] that the forgetful functor of type $\mathbf{Set}^E \to \mathbf{Set}$ which retains only the carrier set of an algebra has a left adjoint. This adjoint maps a set X to the term algebra $c^E_X : E(E^*X) \to E^*X$. Concretely, $c^E_X = [c^m_X]_{m \in E}$ constructs terms of the form $\underline{m}(t_n)_{n \in \mathsf{ar}(m)}$, whereas the adjunction's unit $\eta^E_X : X \to E^*X$ embeds the variables:

$$c^m_X(t_n)_{n \in \mathsf{ar}(m)} := \underline{m}(t_n)_{n \in \mathsf{ar}(m)} \qquad \eta^E_X(x) := \underline{x}$$

The adjuction's counit $\varepsilon^E_\alpha : \langle E^*A, c^E_A \rangle \to \langle A, \alpha \rangle$ evaluates terms under their interpretation $\alpha : EA \to A$:

$$\varepsilon^E_\alpha\big(\underline{m}(t_n)_{n \in \mathsf{ar}(m)}\big) := \alpha^m\big(\varepsilon^E_\alpha(t_n)\big)_{n \in \mathsf{ar}(m)} \qquad \varepsilon^E_\alpha(\underline{a}) := a$$

The preservation of colimits by left adjoints means that the initial object in the category $\mathbf{Set}^E$ is given by $\mu E = \langle E^*\varnothing, c^E_\varnothing \rangle$. Conversely, E^*X can be characterized as the initial algebra

$$[c^E_X, \eta^E_X] \,:\, E(E^*X) + X \,\to\, E^*X$$

for a different endofunctor $Y \mapsto EY + X$. Given an algebra $[\alpha, \rho] : EA + X \to A$, which provides an interpretation $\alpha^m : A^{\mathsf{ar}(m)} \to A$ for each function symbol $m \in E$, and an assignment $\rho : X \to A$ of the variables of X, there is a unique algebra homomorphism $\phi_{\alpha,\rho} : \big\langle E^*X, [c^E_X, \eta^E_X] \big\rangle \to \big\langle A, [\alpha, \rho] \big\rangle$:

$$\begin{tikzcd} E(E^*X) \arrow[r, "c^E_X"] \arrow[d, "E\phi_{\alpha,\rho}"] & E^*X \arrow[d, dashed, "!" ', "\phi_{\alpha,\rho}"] & X \arrow[l, "\eta^E_X"'] \arrow[d, equal] \\ EA \arrow[r, "\alpha"] & A & X \arrow[l, "\rho"'] \end{tikzcd}$$

Note that $\phi_{\alpha,\rho} = \varepsilon_\alpha \circ E^*\rho$ and conversely $\varepsilon_\alpha = \phi_{\alpha,\mathrm{id}_A}$.

This universal property provides a foundation for *effect handlers* [39], a programming language construction which allow a computation to be transformed by reinterpreting its effects and outcome, generalizing the well-established use of exception handlers. Using a different kind F^*Y of computations as the target set, a *handler* $h : E^*X \to F^*Y$ can be specified using the following data:

- a mapping $\rho_h : X \to F^*Y$ which provides a computation $\rho_h(x) \in F^*Y$ meant to be executed when the original computation concludes with a result $x \in X$;
- an interpretation $\alpha_h^m : (F^*Y)^{\mathsf{ar}(m)} \to F^*Y$ for each $m \in E$ providing a computation $\alpha_h^m(k_n)_{n\in\mathsf{ar}(m)}$ to be executed when the original computation triggers the effect m.

Each argument $k_n \in F^*Y$ of α_h corresponds the (recursively tranformed) behavior of the original computation when it is resumed by the outcome $n \in \mathsf{ar}(m)$. We are free to use several of these continuations, each one potentially multiple times, to assign an interpretation to the effect. This flexibility allows handlers to express a great variety of control flow operators found in modern programming languages.

2.4 Final coalgebras

The free monad E^* over an effect signature E allows us to represent *finite* computations with side-effects in E but does not account for *infinite* computations. By considering the coalgebras for $Y \mapsto EX + Y$ instead of algebras, we can construct an alternative monad E^∞ which does not exhibit the same limitation. Coalgebras are pervasive in computer science, where they appear in the guise of automata and transition systems. Their use in the context of algebraic effects therefore presents the additional advantage of establishing a connection with the associated *operational* style of semantics.

Concretely, a coalgebra for the endofunctor $Y \mapsto EY + X$ equips a set of states Q with a transition function $\delta : Q \to EQ + X$ describing what happens when the computation is in a given state $q \in Q$:

- if $\delta(q) = \underline{m}(q'_n)_{n\in\mathsf{ar}(n)}$, the computation triggers the effect $m \in E$, and continues in state q'_n when it is resumed by the outcome $n \in \mathsf{ar}(m)$;
- if $\delta(q) = \underline{x}$, the computation terminates with the result $x \in X$.

We define $\langle E^\infty X, d_X^E\rangle := \nu Y\,.\,EY + X$ as the final such coalgebra, which satisfies the universal property:

$$\begin{tikzcd}
Q \arrow[r, "\delta"] \arrow[d, dashed, "!" ', "\psi_\delta"] & EQ + X \arrow[d, "E\psi_\delta + \mathrm{id}_X"] \\
E^\infty X \arrow[r, "d_X^E"] & E(E^\infty X) + X
\end{tikzcd}$$

An applicable construction of terminal coalgebras can be found in [29].

The action of E^∞ on a function $f : A \to B$ can be defined as $E^\infty f := \psi_{(\mathrm{id}+f)\circ d_A^E}$; the underlying coalgebra $(\mathrm{id}_{E(E^\infty A)} + f) \circ d_A^E : E^\infty A \to E(E^\infty A) + B$ behaves like d_A^E but applies f to any result $x \in A$. The monad's unit $\bar\eta_X^E : X \to E^\infty X$ can be defined as $\bar\eta_X^E := \psi_{\iota_2}$ where the transition system $\iota_2 : X \to EX + X$ immediately terminates, using its state as the outcome. Defining the multiplication $\bar\mu_X^E : E^\infty E^\infty X \to E^\infty X$ involves a coalgebra with states in $E^\infty E^\infty X + E^\infty X$: when states of the outer computation in $E^\infty E^\infty X$ produce a result in $E^\infty X$, we use this result as the new state and switch to executing the inner computation.

This approach has been used to great effect in *interaction trees* [28], a data structure designed along these principles, and formalized using coinductive types in the proof assistant formerly known as Coq. A comprehensive library provides proof principles and categorical combinators for interactions trees, and they are used in the context of the DeepSpec project [12] to interface disparate certified components.

Nevertheless, there are limitations to this approach. In particular, infinite computations often exhibit *silently divergent* behaviors (infinite loops). Modeling these behaviors requires the introduction of a null effect $\tau : \mathbb{1}$ in the signature E, which coalgebras can then use to delay any interaction. While this is feasible, this means the elements of $E^\infty X$ must be considered *up to* τ, in other words in the context of an algebraic theory including the equation $\tau(x) = x$. This requires the use of sophisticated simulation techniques to take into account the distinction between finite iterations (τ^*) and silent divergence (τ^ω).

Less constructively, we can model silent divergence as its own effect $\bot : \varnothing$. We will see in the next section that game semantics can be read as a principled treatment of this approach, which reestablishes a connection with algebras and denotational semantics.

3 Strategies for uninterpreted effects

The theory of algebraic effects has a limited scope: it is intended to be used in conjunction with existing approaches to programming language semantics to facilitate the treatment of computational side-effects. By contrast, game semantics is its own approach to denotational semantics, and game models often feature rich, higher-order compositional structure. On the other hand, the principles underlying their *construction* are somewhat more hazy and a huge variety of models have been proposed. Nevertheless, I begin this section by attempting to give a high-level account of what could be dubbed the *classical* approach, in line with [14, 4, 5, 24].

By reading algebraic signatures as simple games, I then deploy some of the techniques used in game semantics to construct a particularly pleasant model of algebraic effects. This model can be characterized by specializing the theory of algebraic effects to the category $\mathbf{DCPO}_{\bot!}$ of directed-complete pointed partial orders and strict Scott-continuous functions [1]. Notably, the reconciliation operated by game semantics between denotational and operational semantics finds a formal expression in the coincidence between the initial algebras and terminal coalgebras of endofunctors in $\mathbf{DCPO}_{\bot!}$.

3.1 Games and strategies

The games used in game semantics involve two players: the *proponent* P and *opponent* O. The player P represents the *system* being modeled, while O represents its *environment*. The games we will consider are sequential and alternating: the opponent opens the game by playing first, after which the two players contribute every other move.

Traditionally, a game G is specified by a set of *moves* $M_G = M_G^{\mathsf{O}} \uplus M_G^{\mathsf{P}}$ partitioned into opponent and proponent moves. Then *plays* of the game G are finite sequences of the form $m_1 \underline{m_2} m_3 \underline{m_4} \cdots$, where $m_1, m_3, \ldots \in M^{\mathsf{O}}$ are opponent moves and $\underline{m_2}, \underline{m_4}, \ldots \in M^{\mathsf{P}}$ are proponent moves. The set P_G of valid plays of G is often restricted further, to account for the additional structure of the particular game model at hand. In any case, the objects of interest are then the *strategies* for P, which can be modeled as prefix-closed sets of plays $\sigma \subseteq P_G$ which prescribe at most one proponent action in any particular situation:

$$\forall s \in P_G^{\mathrm{odd}} \,.\, \forall \underline{m}, \underline{m}' \in M^{\mathsf{P}} \,.\, s\underline{m},\, s\underline{m}' \in \sigma \;\Rightarrow\; \underline{m} = \underline{m}' \,. \tag{3}$$

Categories of games and strategies can then be constructed. The objects are games. The morphisms are strategies $\sigma : A \to B$ which simultaneously play the game A as the opponent O and the game B as the proponent P, starting with an opening move from the environment in B. Game semantics is related to linear logic [14], and categories of games and strategies often come with a rich structure, for example:

- the game $A \,\&\, B$ is played as A or B at the discretion of the opponent,

- in the game $A \otimes B$, the games A and B are played side by side,
- the game $!A$ allows multiple copies of A to be played at the discretion of the opponent, and
- the game $A^\perp$ reverses the roles of O and P.

There are infinite variations on this basic setup, which have been used to model imperative programming [6], references [3], advanced control structures [31], nondeterminism [23, 20, 42, 37, 16, 27], concurrency [18, 19], etc. Another line of research explores more fundamental variations on constructions of game and strategies [8, 35, 33, 34], attempting to provide simpler models of advanced features and to ground game semantics in a more systematic approach.

3.2 Strategies for effect signatures

Although they are much simpler in structure, effect signatures can be read as games [27]. Under this interpretation, a computation represented as a term in E^*X proceeds in the following way:

- the computation chooses a function symbol $m \in E$,
- the environment chooses an argument $n \in \mathsf{ar}(m)$.

This process is iterated until eventually, the computation chooses a variable $x \in X$ rather than a function symbol, terminating the interaction. In other words, a term $t \in E^*X$ can be interpreted as a strategy for a simple game derived from E and X. We can exploit this analogy to build a model of computations with side-effects which mimics the construction of strategies in game semantics.

Definition 6 (Costrategies over effect signatures)**.** *The* coplays *over an effect signature E with results in a set X are generated by the grammar:*

$$s \in \bar{P}_E(X) ::= \underline{x} \mid \underline{m} \mid \underline{m}ns \qquad (x \in X,\, m \in E,\, n \in \mathsf{ar}(m))$$

The set $\bar{P}_E(X)$ is ordered by a prefix *relation $\sqsubseteq\, \subseteq \bar{P}_E(A) \times \bar{P}_E(A)$, which is the smallest relation satisfying:*

$$\underline{x} \sqsubseteq \underline{x} \qquad\qquad \underline{m} \sqsubseteq \underline{m} \qquad\qquad \underline{m} \sqsubseteq \underline{m}nt \qquad\qquad s \sqsubseteq t \;\Rightarrow\; \underline{m}ns \sqsubseteq \underline{m}nt$$

In addition, the coherence *relation $\frown\, \subseteq \bar{P}_E(X) \times \bar{P}_E(X)$ is the smallest relation satisfying:*

$$\underline{x} \frown \underline{x} \qquad\qquad \underline{m} \frown \underline{m} \qquad\qquad \underline{m} \frown \underline{m}ns \qquad\qquad (n_1 = n_2 \Rightarrow s_1 \frown s_2) \;\Rightarrow\; \underline{m}n_1s_1 \frown \underline{m}n_2s_2$$

Then a costrategy *over the effect signature E with results in X is a downward-closed set $\sigma \subseteq \bar{P}_E(X)$ of pairwise coherent coplays. I will write $\bar{S}_E(X)$ for the set of such costrategies.*

Note that by contrast with the usual convention, the first move is played by the system rather than the environment, hence my use of the terminology *coplays* and *costrategies*. Moreover, formulating the condition (3) by using a coherence relation is slightly non-traditional though not without precedent [15]. Apart from these details, Definition 6 is fairly typical of the game semantics approach.

Switching back to the algebraic point of view, we can interpret the terms of E^*X into $\bar{S}_E(X)$ by defining an algebra $[\alpha, \rho] : E\bar{S}_E(X) + X \to \bar{S}_E(X)$ as follows:

$$\alpha^m(\sigma_n)_{n \in \mathsf{ar}(m)} := \{\underline{m}ns \mid n \in \mathsf{ar}(m), s \in \sigma_n\} \qquad\qquad \rho(x) := \{\underline{x}\}$$

The resulting homomorphism $\phi_{\alpha,\rho} : E^*X \to \bar{S}_E(X)$ is an embedding. However, $\bar{S}_E$ contains many more behaviors, including the undefined or divergent behavior $\varnothing$ as well as infinite behaviors, represented as their sets of finite prefixes. In fact,

Proposition 7. *$\langle \bar{S}_E, \subseteq \rangle$ is a pointed directed-complete partial order.*

Proof. The empty set is trivially a costrategy. For a directed subset $D \subseteq \bar{S}_E(X)$, the union $\bigcup D$ is a costrategy: any two plays $s_1 \in \sigma_1 \in D$ and $s_2 \in \sigma_2 \in D$ must be coherent since $\sigma_1 \cup \sigma_2 \in D \subseteq \bar{S}_E(X)$. □

This invites us to give a characterization for the structure of $\bar{S}_E(X)$ similar to that of E^*X, by working in the category $\mathbf{DCPO}_{\perp !}$ of pointed dcpos and strict Scott-continuous functions.

3.3 Complete partial orders

Directed-complete partial orders (dcpo for short) are used pervasively in denotational semantics of programming languages. Before proceeding further, I summarize a few relevant properties of the category of pointed dcpos and strict Scott-continuous functions.

Definition 8. *A* directed-complete partial order *$\langle A, \sqsubseteq \rangle$ is a partial order with all directed suprema: any directed subset $D \subseteq A$ has a least upper bound $\bigsqcup^{\uparrow} D \in A$, where* directed *means that D is non-empty and that any two $x, y \in D$ have an upper bound $z \in D$. A* pointed *dcpo has a least element $\perp$.*

A strict Scott-continuous map *$f : \langle A, \sqsubseteq \rangle \to \langle B, \leq \rangle$ between pointed dcpos is a function between the underlying sets which preserves directed suprema and $\perp$. The category of pointed dcpos and strict Scott-continous maps is named* $\mathbf{DCPO}_{\perp !}$.

The category $\mathbf{DCPO}_{\perp !}$ is complete and cocomplete, as well as symmetric monoidal closed with respect to the *smash* product. The cartesian product $\prod_{i \in I} \langle A_i, \sqsubseteq_i \rangle$ is as expected: the underlying set $\prod_{i \in I} A_i$ is ordered component-wise and $(\perp_i)_{i \in I}$ is the least element. The smash product $A \otimes B$ is obtained by identifying all tuples of $A \times B$ in which at least one component is $\perp$. The coproduct $\oplus$ is called the *coalesced sum*. It is similar to the coproduct of sets but identifies $\iota_1(\perp_A) = \iota_2(\perp_B) = \perp_{A \oplus B}$.

The *lifting* comonad $(-)_\perp$ associated with the adjunction between $\mathbf{DCPO}_{\perp !}$ and $\mathbf{DCPO}$ extends a dcpo with a new least element $\perp$. Notably it allows to represent (merely) Scott-continuous maps as strict Kleisli morphisms $f : A_\perp \to B$ in $\mathbf{DCPO}_{\perp !}$. Conversely, a strict map out of $A_\perp$ can be specified by its (merely Scott-continuous) action on elements of A. I will use the same notation to describe the flat domain construction $(-)_\perp : \mathbf{Set} \to \mathbf{DCPO}_{\perp !}$, left adjoint to the forgetful functor from $\mathbf{DCPO}_{\perp !}$ to $\mathbf{Set}$.

One remarkable property enjoyed by $\mathbf{DCPO}_{\perp !}$ (and indeed by all $\mathbf{DCPO}_\perp$-enriched categories [9]), is that every enriched endofunctor F has both an initial algebra $c : F\mu F \to \mu F$ and a terminal coalgebra $d : \nu F \to F\nu F$. Furthermore, the two coincide in the sense that $\mu F = \nu F$ and $c^{-1} = d$. This is key to the power of game semantics in deriving *full abstraction* results: it allows strategies to both be defined coinductively and observed inductively, bridging the gap between operational and denotional semantics.

3.4 Algebraic characterization of strategies

The costrategies for an effect signature E and a set of outcomes X can be characterized as

$$\bar{S}_E(X) \cong \mu Y \cdot \hat{E} Y \oplus X_\perp \,, \tag{4}$$

where

Definition 9. *the endofunctor $\hat{E} : \mathbf{DCPO}_{\perp !} \to \mathbf{DCPO}_{\perp !}$ associated with the effect signature E is:*

$$\hat{E} Y := \bigoplus_m \Big(\prod_n Y \Big)_\perp \,.$$

Algebraically, the introduction of $(-)_\perp$ in the definition of $\hat{E}Y$ allows the operations to be non-strict. When an effect $m \in E$ is interpreted, the resulting computation may be partially or completely defined even if the continuation always diverges, in other words it may be the case that $\alpha^m(\perp)_{n \in \mathsf{ar}(m)} \neq \perp$. In terms of game semantics, this corresponds to the fact that all odd-length prefixes of coplays are observed, as witnessed by the case $\underline{m} \in \bar{P}_E$ in Definition 6.

Theorem 10. *For an effect signature E and a set X, the pointed dcpo $\bar{S}_E(X)$ carries the coinciding initial algebra and terminal coalgebra for the endofunctor $Y \mapsto \hat{E}Y \oplus X_\perp$ on $\mathbf{DCPO}_{\perp!}$.*

Proof. The algebra $[\hat{c}^E_X, \hat{\eta}] : \hat{E}(\bar{S}_E X) \oplus X_\perp \to S_E(X)$ can be defined as:

$$\hat{c}^m_X(\sigma_n)_{n \in \mathsf{ar}(m)} := \{\underline{m}ns \mid n \in \mathsf{ar}(m), s \in \sigma_n\} \qquad \hat{\eta}^E_X(x) := \{\underline{x}\}$$

It is easy to verify that the coplays in $\hat{c}^m_X(c_n)_{n \in \mathsf{ar}(m)}$ and $\hat{\eta}^E_X(v)$ are downward closed and pairwise coherent if those of the σ_i's are. The coalgebra $\hat{d}^E_X : \bar{S}_E X \to \hat{E}(\bar{S}_E X) \oplus X_\perp$ can be defined as:

$$\hat{d}^E_X(\sigma) := \begin{cases} \underline{m}(\{s \mid mns \in \sigma\})_{n \in \mathsf{ar}(m)} & \text{if } \underline{m} \in \sigma \\ \underline{x} & \text{if } \underline{x} \in \sigma \\ \perp & \text{otherwise} \end{cases}$$

The coherence condition on σ ensures that the cases are mutually exclusive, and that $\hat{c}^E_X$ and $\hat{d}^E_X$ are mutual inverses. Thanks to the coincidence of initial algebras and terminal coalgebras in $\mathbf{DCPO}_{\perp!}$, this is enough to establish the initiality of $\langle S_E(X), [\hat{c}^E_X, \hat{\eta}^E_X]\rangle$ and the terminality of $\langle S_E(X), \hat{d}^E_X\rangle$. □

While much more general constructions of free algebras in dcpos have been described [13], they tend to be complex. At the cost of a restriction to effect signatures and *sets* of variables, costrategies provide a simple construction with a transparent operational reading. It may also be possible to extend this construction to incorporate limited forms of equational theories by modifying the ordering of coplays.

3.5 Strategies as ideal completions

This algebraic characterization of costrategies given above invites us to consider more closely the relationship between $E^* : \mathbf{Set} \to \mathbf{Set}$ and $\bar{S}_E : \mathbf{Set} \to \mathbf{DCPO}_{\perp!}$. It turns out the costrategies in $\bar{S}_E(X)$ can be constructed as the ideal completion of $E^*(X_\perp)$.

Definition 11. *An* ideal *of a partial order A is a downward closed directed subset of A. I will write $\mathscr{I}A$ for the set of ideals of A, ordered under set inclusion.*

The ideals of A form a dcpo; if A has a least element, then $\mathscr{I}A$ is pointed dcpo. In fact, $\mathscr{I}A$ is the *free* dcpo generated by the partially ordered set A, as expressed by the adjunctions:

$$\begin{tikzcd}
\mathbf{DCPO} \arrow[d, shift right, "(-)_\perp"'] \arrow[d, phantom, "\dashv"] & \mathbf{Pos} \arrow[l, shift right, "\mathscr{I}"'] \arrow[l, phantom, "\perp"] \arrow[d, shift right, "(-)_\perp"'] \arrow[d, phantom, "\dashv"] \\
\mathbf{DCPO}_{\perp!} \arrow[u, shift right, "U"'] \arrow[r, shift right, "U"'] & \mathbf{Pos}_\perp \arrow[l, shift right, "\mathscr{I}"'] \arrow[l, phantom, "\perp"] \arrow[u, shift right, "U"']
\end{tikzcd}$$

As with $(-)_\perp$, a (strict) Scott-continuous map $f : \mathscr{I}A \to B$ can be specified by its (strict) monotonic action on the underlying partial order A. The unit $\downarrow : A \to \mathscr{I}A$ embeds A into the completion.

Definition 12 (Ordering terms). *For an effect signature E and a partial order $\langle X, \leq \rangle$, we extend $\tilde{E}X$ to a partial order $E\langle X, \leq \rangle := \langle EX, \sqsubseteq \rangle$ by defining $\sqsubseteq$ using the rule:*

$$\frac{\forall n \in \mathsf{ar}(m) \cdot x_n \leq y_n}{\underline{m}(x_n)_{n \in \mathsf{ar}(m)} \sqsubseteq \underline{m}(y_n)_{n \in \mathsf{ar}(m)}}$$

Likewise, we extend E^*X to a partial order $\langle E^*X, \sqsubseteq \rangle$ defined using the inductive rules:

$$\frac{\forall n \in \mathsf{ar}(m) \cdot t_n \sqsubseteq t'_n}{\underline{m}(t_n)_{n \in \mathsf{ar}(m)} \sqsubseteq \underline{m}(t'_n)_{n \in \mathsf{ar}(m)}} \qquad \frac{u \leq v}{\underline{u} \sqsubseteq \underline{v}}$$

Theorem 13. *For an effect signature E and a set X, the following partial orders are isomorphic:*

$$\bar{S}_E(X) \cong \mathscr{I}E^*(X_\perp)$$

Proof. It suffices to show that $E^*(X_\perp)$ satisfies the characterization of $\bar{S}_E(X)$ given by Theorem 10. We can proceed in the same way. The algebra $[\hat{c}^E_X, \hat{\eta}^E_X] : \hat{E}\mathscr{I}E^*(X_\perp) \oplus X_\perp \to \mathscr{I}E^*(X_\perp)$ is defined by:

$$\hat{c}^m_X(\downarrow t_n)_{n \in \mathsf{ar}(m)} := \downarrow \underline{m}(t_n)_{n \in \mathsf{ar}(m)} \qquad \hat{\eta}^E_X(x) := \underline{x}$$

The coalgebra $\hat{d}^E_X : \mathscr{I}E^*(X_\perp) \to \hat{E}(\mathscr{I}E^*(X_\perp)) \oplus X_\perp$ can be defined as:

$$\hat{d}^E_X(\downarrow t) := \begin{cases} \underline{m}(\downarrow t_n)_{n \in \mathsf{ar}(m)} & \text{if } t = \underline{m}(t_n)_{n \in \mathsf{ar}(m)} \\ \underline{x} & \text{if } t = \underline{x} \end{cases}$$

As before, it is easy to check that the required conditions hold and that $\hat{c}^E_X$ and $\hat{d}^E_X$ are mutual inverses. □

Remark 14. *This construction could provide a better starting point for incorporating equational theories in strategy models, since it is built from terms rather than coplays.*

Moreover, I believe that $\mathscr{I}$ should generalize to a large class of order completions. This would give a whole spectrum *of models providing support "à la carte" for undefined behaviors, infinite behaviors, and various kinds of nondeterminism up to and including dual nondeterminism [27].*

Making this possible will require a better understanding of the ways initial algebra and terminal coalgebra constructions propagate through the adjunctions defined by order completions, perhaps based on their constructions as limits and colimits of ω-chains [10]. Such an analysis should also shed light on the relationship between the endofunctors $\hat{E}$ and $\tilde{E}$.

4 Algebraic game semantics

The constructions given in the previous section provide a model of algebraic effects grounded in an interpretation of effect signatures as games. Conversely, while effect signatures are a very limited class of games, the analysis we carried out establishes a blueprint for a broader reading of games and strategy constructions under the lens of categorical algebra.

4.1 Multi-sorted signatures

The games described by effect signatures are almost stateless: in essence, the game begins anew every time the system the proponent P is back in control. This is due to the *single-sorted* nature of effect signatures: while *arities* provide the opponent O with different sets of moves in different situations, the single sort does not permit the same flexibility for P. By generalizing the framework to multi-sorted signatures, we gain a considerable amount of expressivity. In fact, multi-sorted signatures can model the mechanics of all sequential alternating games.

Definition 15. *A* multi-sorted effect signature *is a tuple $E = \langle \bar{Q}, \bar{M}, \bar{\delta}, Q, M, \delta \rangle$. The components define:*

- *a set $\bar{Q}$ of* sorts *and a set Q of* arities*;*
- *for every sort $q \in \bar{Q}$ a set $\bar{M}_q$ of* function symbols *and for every $m \in \bar{M}_q$ an arity $\bar{\delta}_q(m) \in Q$;*
- *for every arity $r \in Q$ a set M_r of* argument positions *and for every $n \in M_r$ a sort $\delta_r(n) \in \bar{Q}$.*

This presentation emphasizes the symmetry between the two players. The sorts (proponent state) and arities (opponent states) respectively type operations and argument tuples. The game alternates between a proponent choice of operation and an opponent choice of argument position.

Following the blueprint laid out in §3, we must now assign endofunctors to multi-sorted signatures. We will work in categories of Q or $\bar{Q}$-indexed *tuples* of sets and functions.

Definition 16 (Endofunctors)**.** *Given a multi-sorted effect signature $E = \langle \bar{Q}, \bar{M}, \bar{\delta}, Q, M, \delta \rangle$, we will use:*

$$\hat{E} : \mathbf{Set}^{Q} \to \mathbf{Set}^{\bar{Q}} \qquad (\hat{E}X)_{q \in \bar{Q}} := \sum_{m \in \bar{M}_q} X_{\bar{\delta}_q(m)}$$

$$\check{E} : \mathbf{Set}^{\bar{Q}} \to \mathbf{Set}^{Q} \qquad (\check{E}\bar{X})_{r \in Q} := \prod_{n \in M_r} \bar{X}_{\delta_r(n)}$$

The associated endofunctors can then be defined as:

$$\bar{E} : \mathbf{Set}^{\bar{Q}} \to \mathbf{Set}^{\bar{Q}} \qquad \bar{E} := \hat{E} \circ \check{E}$$

$$E : \mathbf{Set}^{Q} \to \mathbf{Set}^{Q} \qquad E := \check{E} \circ \hat{E}$$

Definition 17 (Term algebra)**.** *Given a multi-sorted effect signature $E = \langle \bar{Q}, \bar{M}, \bar{\delta}, Q, M, \delta \rangle$, and a family of sets $X \in \mathbf{Set}^{\bar{Q}}$, the term algebras $\bar{E}^*X \in \mathbf{Set}^{\bar{Q}}$ and $E^*X \in \mathbf{Set}^{Q}$ are generated by the following grammar:*

$$t \in \bar{E}^*_q ::= \underline{m}k \mid \underline{x} \qquad \left(k \in E^*_{\bar{\delta}_q(m)}, x \in X_q\right)$$

$$k \in E^*_r ::= (t_n)_{n \in M_r} \qquad \left(t_n \in \bar{E}^*_{\delta_r(n)}\right)$$

If the sets X_q are partially ordered, we can define:

$$\frac{k \sqsubseteq k'}{\underline{m}k \sqsubseteq \underline{m}k'} \qquad \frac{x \leq y}{\underline{x} \sqsubseteq \underline{y}} \qquad \frac{\forall n \in M_r \cdot t_n \sqsubseteq t'_n}{(t_n)_{n \in M_r} \sqsubseteq (t'_n)_{n \in M_r}}$$

Note that defining a set of costrategies or strategies requires specifying an initial sort or arity.

Definition 18. *Consider a multi-sorted effect signature $E = \langle \bar{Q}, \bar{M}, \bar{\delta}, Q, M, \delta \rangle$.*

- *The* costrategies *of sort q are the ideals $\bar{\sigma} \in \mathscr{I}\bar{E}^*_q(\varnothing_\perp)_{q \in \bar{Q}}$.*
- *The* strategies *of arity r are the ideals $\sigma \in \mathscr{I}E^*_r(\varnothing_\perp)_{q \in \bar{Q}}$.*

Since the game is entirely described by the multi-sorted signature E, it is no longer necessary to use a non-trivial set of outcomes beyond the undefined outcome $\perp$. It may however be interesting to investigate applications of *strategy variables* by specifying non-empty sets of possible outcomes X^q for each sort q, allowing the strategy to "escape" the game and terminate with an intermediate outcome. It may then be possible to use monadic constructions to define a notion of *sequential* composition of strategies.

4.2 Coinductive games

Consider the *branching* functor $\mathrm{B} : \mathbf{SET} \to \mathbf{SET}$ defined by

$$\mathrm{B}X := \sum_{I \in \mathbf{Set}} X^I .$$

Then the multi-signatures *themselves* can be regarded as coalgebras:

$$E : Q \to \mathrm{BB}Q$$

The terminal coalgebra $\nu\mathrm{BB}$ is a universal multi-signature: every arity in every signature E can be mapped to a proper *game* $G \in \nu\mathrm{BB}$, abstracting away the state-based representation based on sorts and arities.

The branching functor is very versatile. We can think of the shape represented by $\mathrm{B}X$ as the first layer of rooted tree, where X gives the type of subtrees. Multi-layer trees can be obtained by iterating B and the terminal coalgebra can represent infinite trees.

Natural transformations between functors constructed using B represent transformations of layers within these trees, and can be used to manipulate games. These transformations can then be used to "compile" more abstract game models with high-level structure into the more concrete and low-level form based in multi-sorted signatures. This could be used to reduce existing game models to a common framework to better understand and compare their structures.

Lastly, as with strategies, it may be interesting to consider games with variables in X, represented in the terminal coalgebra $\nu Y \cdot \mathrm{BB}Y + X$. Beyond sequential compositionality, variables could be used to introduce fixpoint operators for games and could find applications as "join points" in concurrent game models.

5 Conclusion

Although much work remains to be done, looking at constructions of game models through the prism of categorical algebra offers many promising avenues of investigation. Multi-sorted signatures constitute a low-level representation for sequential alternating games. Looking at existing forms of game semantics using algebraic tools could reveal interesting structures, and suggest principled for the design of general-purpose models capable of accounting for the behaviors of a wide range of heterogeneous components.

Acknowledgments

This work greatly benefited from conversations with Arthur Oliveira Vale, Léo Stefanesco, Paul-André Melliès, and Zhong Shao. It was supported in part by NSF grants 1521523, 1763399, and 2019285.

References

[1] S. Abramsky & A. Jung (1994): *Domain Theory*. In S. Abramsky, D. Gabbay & T. S. E. Maibaum, editors: *Handbook of Logic in Computer Science*, Oxford University Press, pp. 1–168. Available at `http://www.cs.ox.ac.uk/files/298/handbook.pdf`.

[2] Samson Abramsky (2010): *From CSP to Game Semantics*. In: *Reflections on the Work of C.A.R. Hoare*, Springer, London, pp. 33–45, doi:`10.1007/978-1-84882-912-1_2`.

[3] Samson Abramsky, Kohei Honda & Guy Mccusker (1998): *A Fully Abstract Game Semantics for General References*. In: *Proceedings of the Thirteenth Annual IEEE Symposium on Logic in Computer Science*, Society Press, pp. 334–344.

[4] Samson Abramsky & Radha Jagadeesan (1994): *Games and full completeness for multiplicative linear logic*. *J. Symb. Log.* 59(2), pp. 543–574.

[5] Samson Abramsky, Radha Jagadeesan & Pasquale Malacaria (2000): *Full Abstraction for PCF*. *Inf. Comput.* 163(2), pp. 409–470, doi:10.1006/inco.2000.2930. Available at http://www.sciencedirect.com/science/article/pii/S0890540100929304.

[6] Samson Abramsky & Guy McCusker (1997): *Linearity, Sharing and State: A Fully Abstract Game Semantics for Idealized Algol with Active Expressions*. In: *Algol-like Languages*, Birkhäuser, Boston, MA, pp. 297–329, doi:10.1007/978-1-4757-3851-3_10. Available at https://doi.org/10.1007/978-1-4757-3851-3_10.

[7] Samson Abramsky & Guy McCusker (1999): *Game semantics*. In: *Computational logic: Proceedings of the 1997 Marktoberdorf Summer School*, Springer, Berlin, Heidelberg, pp. 1–55, doi:10.1007/978-3-642-58622-4_1.

[8] Samson Abramsky & Paul-André Melliès (1999): *Concurrent Games and Full Completeness*. In: *Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science*, LICS '99, IEEE Computer Society, USA, pp. 431–442, doi:10.1109/LICS.1999.782638.

[9] J. Adamek (1995): *Recursive Data Types in Algebraically ω-Complete Categories*. *Information and Computation* 118(2), pp. 181–190, doi:https://doi.org/10.1006/inco.1995.1061.

[10] Jiří Adámek & Václav Koubek (1979): *Least fixed point of a functor*. *Journal of Computer and System Sciences* 19(2), pp. 163–178.

[11] Arthur Azevedo de Amorim, Nathan Collins, André DeHon, Delphine Demange, Cătălin Hriţcu, David Pichardie, Benjamin C Pierce, Randy Pollack & Andrew Tolmach (2014): *A verified information-flow architecture*. In: *Proceedings of the 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL'14, ACM, New York, NY, USA, pp. 165–178, doi:10.1145/2535838.2535839.

[12] Andrew W Appel, Lennart Beringer, Adam Chlipala, Benjamin C Pierce, Zhong Shao, Stephanie Weirich & Steve Zdancewic (2017): *Position paper: the science of deep specification*. *Phil. Trans. R. Soc. A* 375(2104), p. 20160331, doi:10.1098/rsta.2016.0331.

[13] Ingo Battenfeld (2013): *Comparing approaches to free dcpo-algebra constructions*. *The Journal of Logic and Algebraic Programming* 82(1), pp. 53–70, doi:https://doi.org/10.1016/j.jlap.2012.10.001.

[14] Andreas Blass (1992): *A game semantics for linear logic*. *Ann. Pure Appl. Log.* 56(1–3), pp. 183–220, doi:10.1016/0168-0072(92)90073-9.

[15] Ana C. Calderon & Guy McCusker (2010): *Understanding Game Semantics Through Coherence Spaces*. *Electronic Notes in Theoretical Computer Science* 265, pp. 231–244, doi:https://doi.org/10.1016/j.entcs.2010.08.014. Proceedings of the 26th Conference on the Mathematical Foundations of Programming Semantics (MFPS 2010).

[16] Simon Castellan, Pierre Clairambault, Jonathan Hayman & Glynn Winskel (2018): *Non-angelic concurrent game semantics*. In: *Proceedings of the 21st International Conference on Foundations of Software Science and Computation Structures*, FoSSaCS 2018, Springer, Cham, Switzerland, pp. 3–19, doi:10.1007/978-3-319-89366-2_1.

[17] Joonwon Choi, Muralidaran Vijayaraghavan, Benjamin Sherman, Adam Chlipala & Arvind (2017): *Kami: A Platform for High-Level Parametric Hardware Specification and Its Modular Verification*. *Proc. ACM Program. Lang.* 1(ICFP), doi:10.1145/3110268.

[18] Dan R Ghica & Andrzej S Murawski (2004): *Angelic semantics of fine-grained concurrency*. In: *International Conference on Foundations of Software Science and Computation Structures*, Springer, pp. 211–225.

[19] Dan R. Ghica & Andrzej S. Murawski (2004): *Angelic Semantics of Fine-Grained Concurrency*. In: *Proceedings of the 7th International Conference on Foundations of Software Science and Computation Structures*, FoSSaCS 2004, Springer, Berlin, Heidelberg, pp. 211–225, doi:`10.1007/978-3-540-24727-2_16`.

[20] W. John Gowers & James D. Laird (2018): *A Fully Abstract Game Semantics for Countable Nondeterminism*. In Dan Ghica & Achim Jung, editors: *27th EACSL Annual Conference on Computer Science Logic (CSL 2018)*, *Leibniz International Proceedings in Informatics (LIPIcs)* 119, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp. 24:1–24:18, doi:`10.4230/LIPIcs.CSL.2018.24`. Available at `http://drops.dagstuhl.de/opus/volltexte/2018/9691`.

[21] Ronghui Gu, Jérémie Koenig, Tahina Ramananandro, Zhong Shao, Xiongnan (Newman) Wu, Shu-Chun Weng, Haozhong Zhang & Yu Guo (2015): *Deep Specifications and Certified Abstraction Layers*. In: *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, ACM, New York, NY, USA, pp. 595–608, doi:`10.1145/2676726.2676975`.

[22] Ronghui Gu, Zhong Shao, Hao Chen, Xiongnan Wu, Jieung Kim, Vilhelm Sjöberg & David Costanzo (2016): *CertiKOS: An Extensible Architecture for Building Certified Concurrent OS Kernels*. In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, USENIX Association, Berkeley, CA, USA, pp. 653–669, doi:`10.5555/3026877.3026928`.

[23] Russell Harmer & Guy McCusker (1999): *A fully abstract game semantics for finite nondeterminism*. In: *Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science*, LICS '99, IEEE Computer Society, USA, pp. 422–430, doi:`10.1109/LICS.1999.782637`.

[24] J. M. E. Hyland & C.-H. L. Ong (2000): *On Full Abstraction for PCF: I, II, and III*. *Inf. Comput.* 163(2), pp. 285–408, doi:`10.1006/inco.2000.2917`.

[25] Martin Hyland, Gordon Plotkin & John Power (2006): *Combining effects: Sum and tensor*. *Theoretical Computer Science* 357(1), pp. 70–99, doi:`https://doi.org/10.1016/j.tcs.2006.03.013`. Clifford Lectures and the Mathematical Foundations of Programming Semantics.

[26] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish et al. (2009): *seL4: formal verification of an OS kernel*. In: *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, SOSP '09, ACM, New York, NY, USA, pp. 207–220, doi:`10.1145/1629575.1629596`.

[27] Jérémie Koenig & Zhong Shao (2020): *Refinement-Based Game Semantics for Certified Abstraction Layers*. In: *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '20, ACM, New York, NY, USA, p. 633–647, doi:`10.1145/3373718.3394799`.

[28] Nicolas Koh, Yao Li, Yishuai Li, Li-yao Xia, Lennart Beringer, Wolf Honoré, William Mansky, Benjamin C Pierce & Steve Zdancewic (2019): *From C to interaction trees: specifying, verifying, and testing a networked server*. In: *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*, ACM, pp. 234–248.

[29] Dexter Kozen (2011): *Realization of Coinductive Types*. In Michael Mislove & Joël Ouaknine, editors: *Proc. 27th Conf. Math. Found. Programming Semantics (MFPS XXVII)*, Elsevier Electronic Notes in Theoretical Computer Science, Pittsburgh, PA, pp. 148–155.

[30] Ramana Kumar, Mangnus Myreen, Michael Norrish & Scott Owens (2014): *CakeML: A Verified Implementation of ML*. In: *Proceedings of the 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL'14, ACM, New York, NY, USA, pp. 179–191, doi:`10.1145/2578855.2535841`.

[31] James Laird (1997): *Full abstraction for functional languages with control*. In: *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science*, LICS '97, IEEE Computer Society, USA, pp. 58–67, doi:`10.1109/LICS.1997.614931`.

[32] Xavier Leroy (2009): *Formal Verification of a Realistic Compiler*. *Commun. ACM* 52(7), pp. 107–115, doi:`10.1145/1538788.1538814`.

[33] Paul-André Mellies (2004): *Asynchronous games 2: the true concurrency of innocence*. In: *Proceedings of the 15th International Conference on Concurrency Theory*, CONCUR 2004, Springer, Berlin, Heidelberg, pp. 448–465.

[34] Paul-André Melliès (2019): *Categorical combinatorics of scheduling and synchronization in game semantics*. *Proceedings of the ACM on Programming Languages* 3(POPL), pp. 1–30.

[35] Paul-André Melliès & Samuel Mimram (2007): *Asynchronous Games: Innocence Without Alternation*. In: *Proceedings of the 18th International Conference on Concurrency Theory*, CONCUR 2007, Springer, Berlin, Heidelberg, pp. 395–411, doi:`10.1007/978-3-540-74407-8_27`.

[36] Eugenio Moggi (1991): *Notions of computation and monads*. *Information and computation* 93(1), pp. 55–92.

[37] Andrzej S. Murawski (2008): *Reachability Games and Game Semantics: Comparing Nondeterministic Programs*. In: *Proceedings of the 23rd Annual IEEE Symposium on Logic in Computer Science*, LICS 2008, IEEE Computer Society, USA, pp. 353–363, doi:`10.1109/LICS.2008.24`.

[38] Gordon Plotkin & John Power (2001): *Adequacy for Algebraic Effects*. In: *Proceedings of the 4th International Conference on Foundations of Software Science and Computation Structures*, FoSSaCS 2001, Springer, Berlin, Heidelberg, pp. 1–24, doi:`10.1007/3-540-45315-6_1`.

[39] Gordon Plotkin & Matija Pretnar (2009): *Handlers of algebraic effects*. In: *Proceedings of the 18th European Symposium on Programming*, ESOP 2009, Springer, Berlin, Heidelberg, pp. 80–94, doi:`10.1007/978-3-642-00590-9_7`.

[40] Zhong Shao (2010): *Certified Software*. *Communications of the ACM* 53(12), pp. 56–66, doi:`10.1145/1859204.1859226`.

[41] Věra Trnková, Jiří Adámek, Václav Koubek & Jan Reiterman (1975): *Free algebras, input processes and free monads*. *Commentationes Mathematicae Universitatis Carolinae* 16(2), pp. 339–351. Available at `http://hdl.handle.net/10338.dmlcz/105628`.

[42] Takeshi Tsukada & C.-H. Luke Ong (2015): *Nondeterminism in game semantics via sheaves*. In: *Proceedings of the 30th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS 2015, IEEE Computer Society, USA, pp. 220–231, doi:`10.1109/LICS.2015.30`.

[43] Jianzhou Zhao, Santosh Nagarakatte, Milo MK Martin & Steve Zdancewic (2012): *Formalizing the LLVM intermediate representation for verified program transformations*. In: *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL'12, ACM, New York, NY, USA, pp. 427–440, doi:`10.1145/2103621.2103709`.