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, and strategies for this game constitute the free algebra for the 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 [41].

Over the past decade, researchers have been able to apply this methodology to larger and larger systems: there are now verified compilers [33, 44, 31], operating system kernels [27, 22, 23], and even verified processor designs [12, 18]. As a result, the construction of large-scale, heterogeneous computer systems which are fully verified is now within reach [13]. 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 explore connections between two related but distinct lines of work:

- **Algebraic effects** [39, 40] offer a computational reading of basic concepts in categorical algebra for the purpose of modeling, combining, and reasoning about side-effects in computations. They
are a principled solution grounded in well-established mathematics, and have prompted novel and promising approaches to programming language design.

- **Game semantics** [2, 7] describe interfaces of program components as *games* played between a component and its environment, characterizing the component’s behavior as a *strategy* in this game. This approach has been used 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 to extend this construction to a larger class of games by considering multi-sorted effect signatures.

I will use the notations \( \mathbb{1} := \{\ast\} \) and \( \mathbb{2} := \{\text{tt}, \text{ff}\} \). The set of finite sequences over an alphabet \( \Sigma \) is written \( \Sigma^* \), with \( \varepsilon \) as the empty sequence and \( s \cdot t \) as the concatenation of the sequences \( s \) and \( t \). With that said, 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 present the approach known as *algebraic effects* [39].

### 2.1 Monadic effects

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

\[
\text{abs}(x) := \begin{cases} \text{if } x > 0 \text{ then return } x \text{ else return } -x \\ \end{cases}
\]

(1)

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

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

(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 readbit, print and stop are more difficult to model. Certainly, (2) cannot be described as a function \( g : \mathbb{1} \rightarrow \emptyset \).

The traditional way to address this issue is to capture the available side-effects in a monad \( (T, \eta, \mu) \) [37]. Then \( TX \) represents computations with a result in \( X \) which may also perform side-effects. The monad’s unit \( \eta : X \rightarrow TX \) corresponds to \textit{return}, a pure computation which terminates immediately. The multiplication \( \mu : TTX \rightarrow TX \) first performs the effects of the outer computation, then those of the computation it evaluates to. This allows us to compose the computations \( f : A \rightarrow TB \) and \( g : B \rightarrow TC \) 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 \textit{Set}:

\[
\begin{align*}
TX & := (\Sigma^* \times X_\bot)^2 \\
\eta(x) & := b \mapsto (\varepsilon, x) \\
\mu(i \mapsto (s_i, j \mapsto (s'_{ij}, x_{ij}))) & := b \mapsto (s_b \cdot s'_{bb}, x_{bb})
\end{align*}
\]
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 $\perp$ as well as a result in $X$, the function produces a sequence of characters from a fixed alphabet $\Sigma$. The operations `readbit`, `print` and `stop` can be interpreted as:

- `readbit` $\in \mathcal{T}^2$
- `print` $: \Sigma^* \rightarrow \mathcal{T}^1$
- `stop` $\in \mathcal{T}^\emptyset$

(readbit : $b \mapsto (\varepsilon, b)$)
(print($s$) : $b \mapsto (s, \ast)$)
(stop : $b \mapsto (\varepsilon, \perp)$)

Then, the program (2) can be characterized using the function $g : 1 \rightarrow \mathcal{T}^\emptyset$ defined by:

\[
g(*) := b \mapsto \begin{cases} 
("Hi", \perp) & \text{if } b = \texttt{tt} \\
("Hello", \perp) & \text{if } b = \texttt{ff}
\end{cases}
\]

2.2 Algebraic effects

A long-standing issue with the monadic approach to computational side-effects is that in general, monads do not compose. This makes it difficult to combine programs which use different kinds of side-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 `readbit` : 2, `print[|s|]` : 1 and `stop` : 0. The behavior of the program (2) can then be represented as the term:

\[
\text{readbit}(\text{print["Hi"](stop), print["Hello"](stop))}
\]

and visualized as the tree:

```
readbit
    |   |
    v   v
print["Hi"]   print["Hello"]
    |   |
    v   v
stop   stop
```

Note that `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:

\[
\text{print[|s|]}(\text{print[|s'|]}(x)) = \text{print[|s \cdot s'|]}(x)
\]

can be used to characterize the behavior of the different effects and their possible interactions. Algebraic theories can be combined in various ways \[26\], making possible a compositional treatment of effects.

Below, I present a simple version of the approach, starting with the following notion of effect signature.

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

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

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

The use of arity *sets* allow us to encode effects such as `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 `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 $\text{Set}$.

**Definition 4.** An effect signature $E$ defines an endofunctor $E : \text{Set} \to \text{Set}$ of the same name, as:

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

The elements of $EX$ are terms of depth one with variables in $X$. This is emphasized by the following notation (I use underlining to distinguish term constructors from the underlying elements of $E$):

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

Terms of a fixed depth $k$ can be obtained by iterating the endofunctor as $E^kX$. 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:

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

Note the use of angle brackets $\langle - \rangle$ for simple applications $vs$. parentheses $( - )$ for recursive terms.

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

$$\begin{align*}
EA & \xrightarrow{\alpha} A \\
Ef & \quad f \downarrow \quad f \circ \alpha = \beta \circ Ef \\
EB & \quad \beta \quad \end{align*}$$

It is well-known [42] that the forgetful “carrier set” functor of type $\text{Set}^E \to \text{Set}$ has a left adjoint. This adjoint maps a set $X$ to the term algebra $c^E_X : E(E^*X) \to E^*X$. Concretely, $c^E_X = [c^n_X]_{m \in E}$ constructs terms of the form $\underline{m(t_n)_{n \in \ar(m)}}$, whereas the adjunction’s unit $\eta^E_X : X \to E^*X$ embeds the variables:

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

The adjunction’s counit $e^E_{\alpha} : \langle E^*A, c^E_A \rangle \to \langle A, \alpha \rangle$ evaluates terms under their interpretation $\alpha : EA \to A$:

$$e^E_{\alpha}(\underline{m(t_n)_{n \in \ar(m)}}) := \alpha(\underline{m(e^E_{\alpha}(t_n))_{n \in \ar(m)}}) \quad e^E_{\alpha}(a) := a$$

The monad $\langle E^*, \eta^E, \mu^E \rangle$ arising from this adjunction is called the free monad associated with $E$, and it establishes a connection with the approach described in [subsection 2.1].

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

$$[\underline{c^E_X}, \eta^E_X] : E(E^*X) + X \to E^*X$$
for a different endofunctor \( Y \mapsto EY + X \). Given an algebra \( [\alpha, \rho] : EA + X \to A \), which provides an interpretation \( \alpha^m : A^{ar(m)} \to A \) 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} : \langle E^*X, \{ \eta^X_E, \epsilon^E_{\alpha} \} \rangle \to \langle A, [\alpha, \rho] \rangle : \\
\begin{align*}
E(E^*X) & \xrightarrow{c^E_E} E^*X \xleftarrow{\eta^E_X} X \\
\downarrow E\phi_{\alpha, \rho} & \quad \downarrow \phi_{\alpha, \rho} \\
EA & \xrightarrow{\alpha} A \xleftarrow{\rho} X
\end{align*}

Note that \( \phi_{\alpha, \rho} = \epsilon_a \circ E^* \rho \) and conversely \( \epsilon_a = \phi_{\alpha, \text{id}_X} \).

This universal property provides a foundation for effect handlers [40], a programming language construction which allows 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 \( \alpha_h^m : (F^*Y)^{ar(m)} \to F^*Y \) for each \( m \in E \) providing a computation \( \alpha_h^m(k_n)_{n \in ar(m)} \) to be executed when the original computation triggers the effect \( m \).

Each argument \( k_n \in F^*Y \) of \( \alpha_h \) corresponds the (recursively transformed) behavior of the original computation when it is resumed by the outcome \( n \in ar(m) \). 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 \( Y \mapsto EY + X \) instead of algebras, we can construct an alternative monad \( E^\omega \) which does not exhibit the same limitation. Coalgebras are ubiquitous 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 \( \delta(q) = m\langle q_n \rangle_{n \in ar(m)} \), the computation triggers the effect \( m \in E \), and continues in state \( q_n \) when it is resumed by the outcome \( n \in ar(m) \);
- if \( \delta(q) = x \), the computation terminates with the result \( x \in X \).

We define \( (E^\omega X, d^E_X) := vY.EY + X \) as the final such coalgebra, which satisfies the universal property:

\[
\begin{array}{ccc}
Q & \xrightarrow{\delta} & EQ + X \\
\downarrow \psi_q & & \downarrow d^E_X \\
E^\omega X & \xrightarrow{\eta^E_X} & E(E^\omega X) + X
\end{array}
\]

An applicable construction of terminal coalgebras can be found in [30].
The action of $E^\omega$ on a function $f : A \to B$ can be defined as $E^\omega f := \Psi_{(\id + f) \circ d_E^f}$; the underlying coalgebra $(\id_{E(E^\omega A)} + f) \circ d_E^f : E^\omega A \to E(E^\omega A) + B$ behaves like $d_E^f$ but applies $f$ to any result $x \in A$. The monad’s unit $\eta_E^f : X \to E^\omega X$ can be defined as $\eta_E^f := \psi_t$, where the transition system $t_2 : X \to E^\omega X + X$ immediately terminates, using its state as the outcome. Defining the multiplication $\mu_E^f : E^\omega E^\omega X \to E^\omega X$ involves a coalgebra with states in $E^\omega E^\omega X + E^\omega X$: when states of the outer computation in $E^\omega E^\omega X$ produce a result in $E^\omega X$, 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 [29], a data structure designed along these principles, and formalized using coinductive types in the Coq proof assistant. A comprehensive library provides proof principles and categorical combinators for interactions trees, and they are used in the context of the DeepSpec project [13] 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 $\tau : \bot$ in the signature $E$, which coalgebras can then use to delay any interaction. While this is feasible, this means the elements of $E^\omega X$ must be considered up to $\tau$, 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 ($\tau^*$) and silent divergence ($\tau^\omega$).

Less constructively, we can model silent divergence as its own effect $\bot : \emptyset$. 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. Game models often feature rich, high-order compositional structures, reflecting the languages they are designed to interpret, and the origins of the technique in the semantics of linear logic. On the other hand, the principles underlying their construction are somewhat more hazy and huge variety of approaches 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 [15, 4, 5, 25]. 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 $\text{DCPO}_{\bot!}$ 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 $\text{DCPO}_{\bot!}$.

#### 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 $M_G = M_G^O \cup M_G^P$ partitioned into opponent and proponent moves. Then plays of the game $G$ are finite sequences of the form $m_1 m_2 m_3 m_4 \ldots$, where
Then a costrategy \( J' \) of effective \( J \) is said to be \( \sigma \)-coherent if

\[
\forall s \in P^\sigma_G : \forall m, m' \in M^P : s m, s m' \in \sigma \Rightarrow m = m'.
\]

Although plays are finite, infinite behaviors can be modeled as prefix-closed sets of finite approximations.

Categories of games and strategies can then be constructed. The objects are games. The morphisms are strategies \( \sigma : A \rightarrow B \) 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 \([15]\), 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 \([32]\), nondeterminism \([24, 21, 43, 38, 17, 28]\), concurrency \([19, 20]\), etc. Another line of research explores more fundamental variations on constructions of game and strategies \([8, 36, 34, 35]\), attempting to provide simpler models of advanced features and to ground game semantics in a more systematic approach.

### 3.2 Strategies for effect signatures

Effect signatures can be read as particularly simple games \([28]\). 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 \( n \in \text{ar}(m) \).

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:

\[
s \in \bar{P}_E(X) ::= x \mid m \mid n s \quad (x \in X, m \in E, n \in \text{ar}(m))
\]

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:

\[
x \sqsubseteq x \quad m \sqsubseteq m \quad m \sqsubseteq n m t \quad s \sqsubseteq t \Rightarrow n s \sqsubseteq n t
\]

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

\[
x \odot x \quad m \odot m \quad m \odot n s \quad (n_1 = n_2 \Rightarrow s_1 \odot s_2) \Rightarrow m n_1 s_1 \odot m n_2 s_2
\]

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 \(\{\}\) by using a coherence relation is slightly non-traditional though not without precedent \([16]\). Apart from these details, Definition \([\text{\ref{def:costrategy}}]\) 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 \([\alpha, \rho] : E\bar{S}_E(X) + X \rightarrow \bar{S}_E(X)\) as follows:

\[
\alpha(m(\sigma_n)_{n \in \text{ar}(m)}) := \{mn(s) \mid n \in \text{ar}(m), s \in \sigma_n\} \quad \rho(x) := \{x\}
\]

The resulting homomorphism \(\phi_{\alpha, \rho} : E^*X \rightarrow \bar{S}_E(X)\) is an embedding. However, \(\bar{S}_E\) contains many more behaviors, including the undefined or divergent behavior \(\emptyset\) 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 set \(D\) of costrategies, their union \(\bigcup D\) is again a costrategy. Indeed, since \(D\) is directed, any two plays \(s_1 \in \sigma_1 \in D\) and \(s_2 \in \sigma_2 \in D\) must be coherent: there exists a strategy \(\sigma' \in D\) which includes both \(\sigma_1\) and \(\sigma_2\), hence contains both \(s_1\) and \(s_2\). 

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 \(\text{DCPO} \perp\perp\) of pointed dcpos and strict Scott-continuous functions.

### 3.3 Complete partial orders

Directed-complete partial orders (dcpo for short) are fundamental to 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 \(\sqcup D \in A\), 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 \(\bot\).

A strict Scott-continuous map \(f : \langle A, \sqsubseteq \rangle \rightarrow \langle B, \sqsubseteq \rangle\) between pointed dcpos is a function between the underlying sets which preserves the least element \(\bot\) and all directed suprema. The category of pointed dcpos and strict Scott-continuous maps is named \(\text{DCPO} \perp\perp\).

The category \(\text{DCPO} \perp\perp\) is complete and cocomplete, as well as symmetric monoidal closed with respect to the smash product. The cartesian product \(\prod_{i \in I} A_i\) is ordered component-wise and \((\bot_i)_{i \in I}\) 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 \(\bot\). The coproduct \(A \oplus B\) is called the coalesced sum. It is similar to the coproduct of sets but identifies \(t_1(\bot) = t_2(\bot) = \bot\).

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

One remarkable property enjoyed by \(\text{DCPO} \perp\perp\) (and indeed by all \(\text{DCPO} \perp\perp\)-enriched categories \([9]\)), is that every enriched endofunctor \(F\) has both an initial algebra \(c : F \mu F \rightarrow \mu F\) and a terminal coalgebra \(d : vF \rightarrow FvF\). Furthermore, the two coincide in the sense that \(\mu F = vF\) and \(c^{-1} = d\).
3.4 Algebraic characterization of strategies

The costrategies for an effect signature $E$ and a set of outcomes $X$ can be characterized as
\[ \hat{S}_E(X) \cong \mu Y \cdot \hat{E}Y \oplus X_\bot, \] (4)
where

**Definition 9.** The endofunctor $\hat{E} : \text{DCPO}_{\bot!} \to \text{DCPO}_{\bot!}$ associated with the effect signature $E$ is:
\[ \hat{E}Y := \bigoplus_m \left( \prod_n Y \right)_\bot. \]

Algebraically, the introduction of $(-)_\bot$ 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 $\alpha^m(\bot)_{n \in \text{ar}(m)} \neq \bot$. In terms of game semantics, this corresponds to the fact that all odd-length prefixes of coplays are observed, as witnessed by the case $m \in \hat{P}_E$ in Definition 6.

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

*Proof.* The algebra $[\hat{c}_X^E, \hat{\eta}_X^E] : \hat{E} \hat{S}_E(X) \oplus X_\bot \to \hat{S}_E(X)$ can be defined as:
\[ \hat{c}_X^E(m(\sigma_n)_{n \in \text{ar}(m)}) := \{ ms \mid n \in \text{ar}(m), s \in \sigma_n \} \quad \hat{\eta}_X^E(x) := \{ x \} \]
It is easy to verify that the coplays in $\hat{c}_X^E(m(\sigma_n)_{n \in \text{ar}(m)})$ and $\hat{\eta}_X^E(x)$ are downward closed and pairwise coherent if the $\sigma_i$’s are. The coalgebra $\hat{d}_X^E : \hat{S}_E(X) \to \hat{E} \hat{S}_E(X) \oplus X_\bot$ can be defined as:
\[ \hat{d}_X^E(\sigma) := \begin{cases} m(\{ s \mid ms \in \sigma \})_{n \in \text{ar}(m)} & \text{if } m \in \sigma \\ x & \text{if } x \in \sigma \\ \bot & \text{otherwise} \end{cases} \]
The coherence condition on $\sigma$ ensures that the cases are mutually exclusive and that $\hat{c}_X^E$ and $\hat{d}_X^E$ are mutual inverses. Thanks to the coincidence of initial algebras and terminal coalgebras in $\text{DCPO}_{\bot!}$, this is enough to establish the initiality of $(\hat{S}_E(X), [\hat{c}_X^E, \hat{\eta}_X^E])$ and the terminality of $(\hat{S}_E(X), \hat{d}_X^E)$. $\Box$

While much more general constructions of free algebras in dcpos have been described [14], 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 acting on the ordering of coplays.

3.5 Strategies as ideal completions

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

**Definition 11.** An ideal of a partial order $A$ is a downward closed directed subset of $A$. I will write $\mathcal{I} A$ 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 \( \mathcal{I} A \) is pointed dcpo. In fact, \( \mathcal{I} A \) is the free dcpo generated by the partially ordered set \( A \), as expressed by the adjunctions:

\[
\begin{array}{c}
\text{DCPO} \quad \downarrow \quad \mathcal{I} \quad \uparrow \quad \text{Pos} \\
\downarrow \quad \downarrow U \quad \uparrow \quad \uparrow U \\
\text{DCPO} \quad \downarrow \quad \mathcal{I} \quad \uparrow \quad \text{Pos} \\
\end{array}
\]

The unit \( \downarrow : A \to \mathcal{I} A \) embeds a partial order \( A \) into its completion. A (strict) Scott-continuous map of type \( f : \mathcal{I} A \to B \) can be specified by its (strict) monotonic action \( f(\downarrow a) \) on the elements \( a \in A \).

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

\[
\forall n \in \text{ar}(m) \cdot x_n \leq y_n \\
\quad \frac{\exists x \in \text{ar}(m) \cdot x \leq x^*_n}{m(x_n)_{n \in \text{ar}(m)} \subseteq m(y_n)_{n \in \text{ar}(m)}}
\]

If \( \langle X, \leq \rangle \) has a least element \( \bot \), we extend \( E^*X \) to a partial order \( \langle E^*X, \subseteq \rangle \) using the inductive rules:

\[
\forall n \in \text{ar}(m) \cdot t_n \subseteq t'_n \\
\quad \frac{\exists x \in \text{ar}(m) \cdot t \subseteq t'}{m(t_n)_{n \in \text{ar}(m)} \subseteq m(t'_n)_{n \in \text{ar}(m)}}
\]

Here the elements of \( E^*X \) are interpreted as partial terms, where the special variable \( \bot \in X \) indicates a lack of information about a particular subterm. A situation where \( t_1, t_2 \subseteq t \) denotes that \( t_1 \) and \( t_2 \) are both truncated versions of the more defined term \( t \) and are therefore compatible in the following sense: although \( t_1 \) may be defined where \( t_2 \) is not and vice versa, they will not conflict on any of their defined subterms and they can be merged into \( t_1 \sqcup t_2 \). By using the ideal completion, we can extend this procedure to arbitrary directed sets, enabling the construction of infinite terms.

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

\[
\tilde{\mathcal{S}}_E(X) \cong \mathcal{I} E^*(X_\bot)
\]

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

\[
c^E_X(m(\downarrow t_n)_{n \in \text{ar}(m)}) := \downarrow m(t_n)_{n \in \text{ar}(m)} \quad \hat{\eta}^E_X(x) := \downarrow x
\]

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

\[
\hat{d}^E_X(\downarrow m(t_n)_{n \in \text{ar}(m)}) := m(\downarrow t_n)_{n \in \text{ar}(m)} \quad \hat{d}^F_X(\downarrow x) := x
\]

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

Finally, it has been shown [11] that \( \mathcal{I} E^*(X_\bot) \cong E^\omega(X_\bot) \). Hence, Theorem 13 also establishes a connection between strategies and the coalgebraic approach discussed in §2.4.
Remark 14. The construction of strategies as ideal completions could provide a better starting point for incorporating equational theories in strategy models, since it is built from terms rather than plays.

Moreover, I believe that \( \mathcal{F} \) 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 [28].

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 \( \omega \)-chains [10]. Such an analysis should also shed light on the relationship between the endofunctors \( \hat{E} \) and \( E \).

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 above 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 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 \( E = (\bar{Q}, \bar{M}, \bar{\delta}, Q, M, \delta) \). 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 \( m \in M_q \) 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 \( \bar{\delta}_r(n) \in \bar{Q} \).

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 \( E = (\bar{Q}, \bar{M}, \bar{\delta}, Q, M, \delta) \), we will use:

\[
\hat{E} : \text{Set}^{\bar{Q}} \to \text{Set}^{\bar{Q}} \\
(\hat{E}X)_{q \in \bar{Q}} := \sum_{m \in M_q} X_{\bar{\delta}_q(m)}
\]

\[
\check{E} : \text{Set}^{\bar{Q}} \to \text{Set}^{Q} \\
(\check{E}X)_{r \in Q} := \prod_{n \in M_r} X_{\bar{\delta}_r(n)}
\]

The associated endofunctors can then be defined as:

\[
\hat{E} : \text{Set}^{\bar{Q}} \to \text{Set}^{\bar{Q}} \quad \check{E} := \hat{E} \circ \hat{E}
\]

\[
\hat{E} : \text{Set}^{\bar{Q}} \to \text{Set}^{Q} \quad \check{E} := \check{E} \circ \check{E}
\]
Definition 17 (Term algebra). Given a multi-sorted effect signature $E = (\bar{Q}, \bar{M}, \bar{\delta}, Q, M, \delta)$, and a family of sets $X \in \text{Set}^{\bar{Q}}$, the term algebras $\bar{E}^* X \in \text{Set}^{\bar{Q}}$ and $E^* X \in \text{Set}^{Q}$ are generated by the following grammar:

$$
t \in \bar{E}^* := m \cdot k \mid x \quad (k \in E^*_q) \quad x \in X_q
$$

$$
k \in E^*_r := (t_n)_{n \in M_r} \quad (t_n \in \bar{E}^*_r) \quad (n \in M_r)
$$

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

$$
k \sqsubseteq k' \quad x \leq y \quad \forall n \in M_r \cdot t_n \sqsubseteq t'_n \quad (n \in M_r)
$$

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

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

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

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 $\bot$. 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 $B : \text{SET} \to \text{SET}$ defined by

$$
B(X) := \sum_{I \in \text{Set}} X^I.
$$

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

$$
E : Q \to \nu B
$$

The terminal coalgebra $\nu BB$ is a universal multi-signature: every arity in every signature $E$ can be mapped to a proper game $G \in \nu BB$, 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 $BX$ 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 $\nu Y \cdot BB + X$. 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 general principles 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, Léon Stefanesco, Paul-André Melliès, and Zhong Shao. It was supported in part by NSF grants 1521523, 1763399, and 2019285.

References


