Algebraic Effects
& their Applications to Cryptography

Niels Voorneveld

March 22, 2024

Workshop on Process Theory for Security Protocols and Cryptography, Tallinn 2024

CYBERNETICA

Towards Denotational Semantics

Features of Cryptographic Protocols:

- Calling operations such as encode and decode.
- Invoking a plethora of effects: Probability, adversarial nondeterminism, global state, time, input-output.

Goal: Formulate a denotational semantics for cryptographic libraries and protocols.

Outline:

- How do we describe algebraic effects.
- How do we describe handling algebraic effects.
- Framework for describing complex interactions.

Effectful Programs

Equality and Equivalence

Algebraic effects are about operations and *equations*.

It facilitates establishing *equivalence* between effectful programs.

Two programs are equivalent if there is no observable difference between them.

Use cases:

- This implemented program has no observable difference from the specifications.
- This update does not change the program in an observable way.

Side note: Generalizing to *inequations* allows one to talk about program improvements as well.

Algebras of Programs

The standard observable behaviour of a program is its output, and equivalence is defined based on whether programs produce the same output.

A program may however be *effected* by its environment, which changes the output

A basic situation:

- A program calls an operation of type $A \to B$, providing an argument of type A.
- The environment answers, returning an element of type B.
- The program continues, dependent on the returned element.

The output of the program depends on the answers of the environment, which may be difficult to predict.

Algebraic Operations

A program of type X calling an *algebraic operation* $\mathrm{op} : A \to B$ needs to specify an argument $a : A$ and a continuation $B \to X$.

$$\frac{a : A \qquad f : B \to X}{\mathtt{op}_a(f) : X}$$

If B has only n elements, we can see f as a tuple $(x_1, \ldots, x_n)$ of X, and write $\mathtt{op}_a(x_1, \ldots, x_n)$.

A set of operations S is a signature, or container.

Given a signature S, and a set of outputs Y, we write $T_S Y$ as the minimal set of terms closed under operations of S and which returns values of Y:

- For each $y \in Y$, $\mathtt{return}(y) \in T_S Y$.
- For each $\mathrm{op} : A \to B \in S$, $a \in A$ and $f : B \to T_S Y$, $\mathtt{op}_a(f) \in T_S Y$.

Example: Coin toss

Consider a coin toss operations $\mathtt{toss} : 1 \to 2$ accepting no input and returning a bit.

$T_{\{\mathtt{toss}\}} Y$ are binary trees whose leaves are labeled by Y.

Equations:

- $\mathtt{toss}(x, x) = x$.
- $\mathtt{toss}(x, y) = \mathtt{toss}(y, x)$.
- $\mathtt{toss}(\mathtt{toss}(x, y), \mathtt{toss}(z, w)) =$ $\mathtt{toss}(\mathtt{toss}(x, z), \mathtt{toss}(y, w))$.

Example: State

A set of states M.
Operations:

- $\mathtt{update} : M \to 1$.
- $\mathtt{lookup} : 1 \to M$.

Equations:

- $\mathtt{update}_a(\mathtt{update}_b(x)) = \mathtt{update}_b(x)$.
- $\mathtt{update}_a(\mathtt{lookup}(f)) = \mathtt{update}_a(f(a))$.
- $\mathtt{lookup}(i \mapsto \mathtt{update}_i(f(i))) = \mathtt{lookup}(f)$.
- $\mathtt{lookup}(i \mapsto x) = x$.

Example: Encode

Set of plaintexts P, cryphertext C and keys K.

Operations:

- $\texttt{encode} : K \times P \to C$.
- $\texttt{decode} : K \times C \to P$.

Equation:

$$\texttt{encode}_{k,p}(c \mapsto \texttt{decode}_{k,c}(f)) = f(p)$$

Environments

Co-Operations

An environment of type Y resolving an *algebraic operation* $\mathtt{op} : A \to B$ needs to specify an answer $b : A$ and a continuation Y.

$$\frac{a : A \qquad e : Y}{\mathtt{op}_a(e) : B \times Y}$$

Given a set of operations S, and a set of states Y, we write $D_S Y$ as the maximal set of response patterns closed under the following operations

- For each $e \in D_S Y$, $\mathtt{return}(e) \in Y$.
- For each $e \in D_S Y$, $\mathtt{op} : A \to B \in S$, and $a \in A$, $\mathtt{op}_a(e) \in B \times D_S Y$.

Co-Equations

One way to specify environment behaviour is with *co-equations*.

A co-equations specifies a property of behaviour an environment should adhere to (or avoid).

It tells us whether different ways of extracting data and manipulating the environment are equal.

Examples:

- Determinism: The environment will always give the same answer to the same question.
- Immutability: The environment will not change its internal state when asked certain questions.
- An adversarial environment will behave according to some rules.

Coalgebraic Specification

A complementary way of specifying environment behaviour is by defining explicit answers dependent on an internal state.

An element of $D_S Y$ can be specified in the following way:

- A set of internal states K.
- For each $(\mathrm{op} : A \to B) \in S$ a function $K \times A \to K \times B$.

This gives a (comonad) coalgebra $\beta : K \to D_S K$.

Together with the following, we get an element of $D_S Y$:

- An initial state $k_0 \in K$.
- An output function $K \to Y$.

Example: Seeds

Consider again the coin toss operation $\mathtt{toss} : 1 \to 2$.

Consider a set of seeds S, and a function $\mathrm{pop} : S \to 2 \times S$ which draws a bit and changes the seed.

- S could be $2^{\mathbb{N}}$.
- It could be some pseudo random number generator.

$\mathrm{pop} : S \times 1 \to 2 \times S$ together with an initial seed $s_0 \in S$ defines an element in $D_{\{\mathtt{toss}\}} S$.

Example: Memory

A set of states M with operations:

- $\texttt{update} : M \to 1$.
- $\texttt{lookup} : 1 \to M$.

Some coequations:

- $\texttt{update}_a(\texttt{update}_b(x)) = \texttt{update}_a(x)$.
- $\texttt{lookup}(\texttt{update}_a(x)) = (a, \texttt{update}_a(x))$.
- $\texttt{lookup}(\texttt{lookup}(e)) = (a, (a, e'))$ where $(a, e') = \texttt{lookup}(e))$.

We define resolution using an initial state $m_0 \in M$ and functions:

- $\texttt{update}' : M \times M \to M \times 1$, where $\texttt{update}'(m, a) = (a, *)$
- $\texttt{lookup}' : M \times 1 \to M \times M$, where $\texttt{lookup}'(m, *) = (m, m)$.

Example: Encode

Set of plaintexts P, cryphertext C and keys K.

Operations:

- $\texttt{encode} : K \times P \to C$.
- $\texttt{decode} : K \times C \to P$.

Co-equations:

- $\texttt{decode}_{k,c}(e') = (p, e)$, for $(c, e') = \texttt{encode}_{k,p}(e)$.
- (Determinism) $\texttt{encode}_{k,p}(e') = (c, e'')$ for $\texttt{encode}_{k,p}(e)(c, e')$.

Effectful Environments

Invoking Further Effects

Given two sets of operations S and Z, and a set of states Y, we write $D_S^Z Y$ as the maximal set of response patterns closed under the following operations

- For each $e \in D_S^Z Y$, $\texttt{return}(e) \in Y$.
- For each $e \in D_S^Z Y$, $\texttt{op} : A \to B \in S$, and $a \in A$, $\texttt{op}_a(e) \in T_Z(B \times D_S Y)$.

Co-equations on S now also consider equations on Z (relatively unexplored territory).

Coalgebraic specifications now also invoke effects from Z.

Effectful Coalgebraic Specification

Given containers S and Z:

$$D_S^Z Y = \nu X. Y \times \Pi_{(\mathrm{op}:A \to B) \in S} A \to (B \times T_Z X)$$

An element e of $D_S^Z Y$ can be specified in the following way:

- A set of internal states K.
- For each $(\mathrm{op} : A \to B) \in S$ a function $\mathrm{op}^e : K \times A \to T_Z(K \times B)$.

This gives a (comonad) coalgebra $\beta : K \to D_S^Z K$.

Together with the following, we get an element of $D_S^Z Y$:

- An initial state $k_0 \in K$.
- An output function $K \to Y$.

Example: Encode Subprotocol

Set of plaintexts P, cryphertext C and keys K.

Operations of S	Operations of Z
$\texttt{encodeS} : P \to C$	$\texttt{encodeZ} : K \times P \to C$
$\texttt{decodeS} : C \to P$	$\texttt{decodeZ} : K \times C \to P$
	$\texttt{random} : 1 \to K$

Let states be $M = K_\perp = K \cup \perp$, and define

- $\overline{\texttt{encodeS}} : M \times P \to M \times C$,
  $\overline{\texttt{encodeS}(k, p)} = \texttt{encodeZ}_{k,p}(c \mapsto (k, c))$,
  $\overline{\texttt{encodeS}(\perp, p)} = \texttt{random}(k \mapsto \texttt{encodeZ}_{k,p}(c \mapsto (k, c)))$.
- $\overline{\texttt{decodeS}} : M \times C \to M \times P$, similarly.

With initial $\perp \in M$, this gives an element of $D_S^Z(M)$.

Interactions

Programs asking questions from a signature S:

$$T_S Y = \mu X.Y + \Sigma_{(\mathrm{op}:A \to B) \in S} A \times (B \to X)$$

$$D_S Y = \nu X.Y \times \Pi_{(\mathrm{op}:A \to B) \in S} A \to (B \times X)$$

$$D_S^Z Y = \nu X.Y \times \Pi_{(\mathrm{op}:A \to B) \in S} A \to T_Z(B \times X)$$

Programs interact with environments:

$$T_S X \times D_S Y \to X \times Y$$

$$T_S X \times D_S^Z Y \to T_Z(X \times Y)$$

Systems and Protocols

Category of Containers

Objects are given by containers S in Set.

A morphism $m : S \to Z$ is given by:

- A state space K_m.
- An initial state $k_m \in K_m$.
- For each $(\mathrm{op} : A \to B) \in S$ a function $\mathrm{op}_m : A \times K_m \to T_Z(B \times K_m)$.

A morphism m from S to Z specifies:

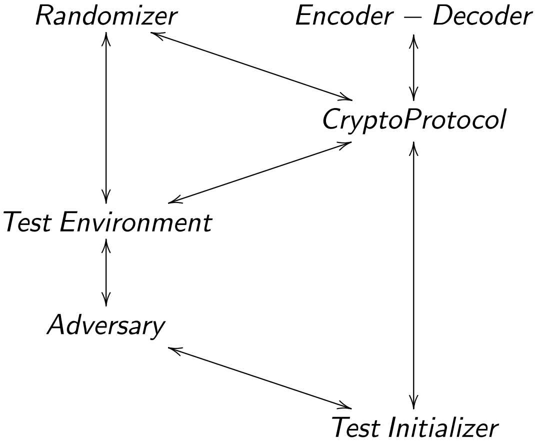
- A monad morphism from T_S to T_Z.
- A comonad morphism from D_Z to D_S.
- An element of $D_S^Z K_m$.
- A runner $\rho_m : T_S X \times K_m \to T_Z(X \times K_m)$.

Gives Symmetric monoidal with coproduct over objects.

Crypto Protocol

$$
\begin{tikzcd}
\mathit{Randomizer} \arrow[dr, leftrightarrow] & & \mathit{Encoder} - \mathit{Decoder} \arrow[dl, leftrightarrow] \\
& \mathit{Crypto\ Protocol} \arrow[d, leftrightarrow] & \\
& \mathit{User} &
\end{tikzcd}
$$

Testing Security



Communication

$$
\begin{tikzcd}
\mathit{Encoder} - \mathit{Decoder} \arrow[d, leftrightarrow] & & & & \mathit{Encode} - \mathit{Decode} \arrow[d, leftrightarrow] \\
\mathit{Alice} \arrow[drr, leftrightarrow] & & & & \mathit{Bob} \arrow[dll, leftrightarrow] \\
& & \mathit{Scheduler} & &
\end{tikzcd}
$$

Final thoughts

Equations

Adversarial decisions can be modeled using demonic nondeterminism: worst case scenario is assumed.

Testing security of a particular scheme then composes into a combination of probability and demonic nondeterminism.

Have a category which limits to polynomial time computations:

- Sized objects, with time monad.
- Probability, quotiented over negligibility.

Free Cornering

The monad-comonad interaction model always has a root:
An active component leading the interaction.

The *free cornering* model can be used to extend the framework, allowing protocols with multiple active parties.

E.g. Both tester and adversary are active, but could await others' actions.

There is a "functor" from the category of containers over Set into the category of horizontal morphisms in the free cornering with choice and iteration over Set.

References

Tarmo Uustalu and N.V.: Algebraic and coalgebraic perspectives on interaction laws, Proc. of 18th Asian Symp. on Programming Languages and Systems, APLAS 2020, to appear in LNCS

Chad Nester, N.V.: Protocol Choice and Iteration for the Free Cornering, Journal of Logical and Algebraic Methods in Programming, JLAMP, Volume 137

Fin