THE ORIGINS OF FREE

Adam Warski, SoftwareMill
27 May 2017, LambdaConf

THE PLAN

- Why bother?
- Universal algebra
- Free algebras
- The meaning of ~~life~~ free
- Monads & free monads
- Free monads in Haskell, Cats and Scalaz

- It’s simpler than it sounds

WHY BOTHER WITH FREE?

- *Free monads:*
  - programs as values
  - separate definition from interpretation
  - testing
  - composability
  - use a high-level language
    - hide low-level concerns

WHAT IS AN ALGEBRA?

“algebra is the study of mathematical symbols and the rules for manipulating these symbols"
Wikipedia, 2017

“the part of mathematics in which letters and other general symbols are used to represent numbers and quantities in formulae and equations."
Google Search, 2017

```
y = ax + b

E = mc²

f(10◊x) = K(▸9)

def sum(l: List[L]) = l.fold(_ + _)

main = getCurrentTime >>= print
```

UNIVERSAL ALGEBRA: SIGNATURE

- Goal: Model programs as algebras
- Let's generalise!
- Studies algebraic structures, rather than concrete models
- Syntax: algebraic signature $\Sigma = (S, \Omega)$
  - type names: set S
  - operation names: family Ω of sets indexed by $S^* \times S$

UNIVERSAL ALGEBRA: SIGNATURE EXAMPLE

$$
\begin{aligned}
S &= \{int, str\} \\
\Omega_{\epsilon,int} &= \{0\} \\
\Omega_{int,int} &= \{succ\} \\
\Omega_{(int,int),int} &= \{+\} \\
\Omega_{(str,str),str} &= \{++\} \\
\Omega_{int,str} &= \{toString\}
\end{aligned}
$$

$$toString(succ(0) + succ(succ(0)))$$

UNIVERSAL ALGEBRA: SIGNATURE EXAMPLE

$$
\begin{aligned}
S &= \{v\} \\
\Omega_{\epsilon,v} &= \{1\} \\
\Omega_{(v,v),v} &= \{*\}
\end{aligned}
$$

$$(1 * 1) * (1 * (1 * 1))$$

UNIVERSAL ALGEBRA: ALGEBRA

- A specific interpretation of the **signature**
  - for each type, a set
  - for each operation, a function between appropriate sets

$\Sigma = (S, \Omega)$, $S = \{int, str\}$ and $\Omega = \{0, succ, +, ++, toString\}$

We can define a Σ-algebra A:

$|A|_{int} = \{0, 1, 2, ...\} = \mathbb{N}$
$|A|_{str} = \{"a", "aa", ..., "b", "ab", ...\}$

$succ_A = \lambda x.x + 1$
$+_A = \lambda xy.x + y$
$\ldots$

UNIVERSAL ALGEBRA: ALGEBRA

- As a side-note, F-algebras:

```
type Algebra[F[_], A] = F[A] => A
// or
type Algebra f a = f a -> a
```

are a generalisation of single-sorted algebras presented here

TERM ALGEBRA

- Can we build an algebra out of pure syntax?
- Expressions (terms) that can be built from the signature
- Rather boring, no interpretation at all

$\Sigma = (S, \Omega),\ S = \{int, str\}$ and $\Omega = \{0, succ, +, ++, toString\}$

We define the term algebra T_Σ:

$$|T_\Sigma|_{int} = \{0, succ(0), succ(succ(0)), ..., 0 + 0, 0 + succ(0), ...\}$$
$$|T_\Sigma|_{str} = \{toString(0), toString(succ(0)), ..., toString(0) ++ toString(0), ...\}$$

$succ_{T_\Sigma}(t) = succ(t)$, e.g. $succ_{T_\Sigma}(succ(0)) = succ(succ(0))$
$+_{T_\Sigma}(t_1, t_2) = t_1 + t_2$
...

TERM ALGEBRA

$\Sigma = (S, \Omega),\ S = \{int, str\}$ and $\Omega = \{0, succ, +, ++, toString\}$

- Defined **inductively**
  - base: all constants are terms
  - step: any functions we can apply on previous terms

$\{0\}$
$\{0, 0 + 0, succ(0), toString(0)\}$
$\{0, 0 + 0, succ(0), 0 + succ(0), succ(0) + 0, succ(0) + succ(0),$
$toString(0), toString(succ(0)), toString(0) + +toString(0)\}$

TERM ALGEBRA

$\Sigma = (S, \Omega)$, $S = \{v\}$ and $\Omega = \{1, *\}$

$1, 1 * 1, (1 * 1) * 1, 1 * (1 * 1), 1 * ((1 * 1) * 1), ...$

HOMOMORPHISM

- Homomorphism is a function between algebras
  - For each type, functions between type interpretations
  - Such that operations are preserved

$\Sigma = (S, \Omega)$, $S = \{int, str\}$ and $\Omega = \{0, succ, +, ++, toString\}$

When A and B are Σ-algebras, $f : A \to B$ is a homomorphism when:
$f_{int} : |A|_{int} \to |B|_{int}$
$f_{str} : |A|_{str} \to |B|_{str}$

$\forall_{x \in |A|_{int}} f_{int}(succ_A(x)) = succ_B(f_{int}(x))$
$\forall_{xy \in |A|_{int}} f(x +_A y) = f(x) +_B f(y)$
$\forall_{x \in |A|_{int}} f_{str}(toString_A(x)) = toString_B(f_{int}(x))$

INITIAL ALGEBRA

Σ-algebra I is **initial** when for *any other* Σ-algebra A there is **exactly one** homomorphism from I to A.

Theorem 1 T_Σ *is initial*

INITIAL ALGEBRA

Theorem 1 T_Σ *is initial*

$\Sigma = (S, \Omega)$, $S = \{int, str\}$ and $\Omega = \{0, succ, +, ++, toString\}$

We can define a Σ-algebra A:

$|A|_{int} = \{0, 1, 2, ...\} = \mathbb{N}$

$|A|_{str} = \{"a", "aa", ..., "b", "ab", ...\}$

$succ_A = \lambda x.x + 1$

$+_A = \lambda xy.x + y$

$\dots$

$$f : T_\Sigma \to A$$

$f(0_{T_\Sigma}) = 0_A$

$f(succ_{T_\Sigma}(t)) = succ_A(f(t))$

...

INITIAL ALGEBRA

Σ-algebra I is **initial** when for *any other* Σ-algebra A there is **exactly one** homomorphism between them.

Theorem 1 T_Σ *is initial*

- Only one way to interpret a term
- *no junk*: term algebra contains only what's absolutely necessary
- *no confusion*: no two values are combined if they don't need to be
- There's only one initial algebra (up to isomorphism)

INITIAL ALGEBRA

- This algebra is definitely **not initial**:

  $\Sigma = (S, \Omega),\ S = \{int, str\}$ and $\Omega = \{0, succ, +, ++, toString\}$

  We can define a Σ-algebra A:

  $|A|_{int} = \{0, 1, 2, ...\} = \mathbb{N}$
  $|A|_{str} = \{"a", "aa", ..., "b", "ab", ...\}$

- *Junk*: strings *"a", "b", ...*
- *Confusion: 0+succ(0)* is same as *succ(0)+0*

TERMS WITH VARIABLES

For any set X, $T_\Sigma(X)$ is the term algebra with X added as "constants"
(but called variables)

$\Sigma = (S, \Omega)$, $S = \{int, str\}$ and $\Omega = \{0, succ, +, ++, toString\}$

$$X_{int} = \{i, j, k\}$$
$$X_{str} = \{s_1, s_2\}$$

$$succ(i) + j + succ(succ(k))$$
$$s_1 + +toString(0)$$
$$toString(succ(0) + k) + +s_2$$

TERMS WITH VARIABLES

For any set X, $T_\Sigma(X)$ is the term algebra with X added as "constants"
(but called variables)

$\Sigma = (S, \Omega)$, $S = \{v\}$ and $\Omega = \{1, *\}$

$$X_v = \{x, y, z, ...\}$$

$$1 * x$$
$$x * (y * z)$$
$$(x * y) * z)$$
$$(x * 1) * (y * (1 * 1))$$

FREE ALGEBRA

For any set X, $T_\Sigma(X)$ is the term algebra with X added as "constants"
(but called variables)

Σ-algebra I is **free over X** ($X \subset I$) when for *any other*
Σ-algebra A, *any function* $f : X \to |A|$ **extends uniquely**
to a homomorphism $f^\# : I \to A$ between them.

Theorem 1 *For any variable set* X, $T_\Sigma(X)$ *is free*

- An interpretation of the variables determines an interpretation of any term

FREE ALGEBRA EXAMPLE

$\Sigma = (S, \Omega),\ S = \{int, str\}$ and $\Omega = \{0, succ, +, ++, toString\}$

$X_{int} = \{i, j, k\}$
$X_{str} = \{s_1, s_2\}$
$|A|_{int} = \mathbb{N},\ |A|_{str} = \{"a", "aa", ..., "b", "ab", ...\}$

$succ_A = \lambda x.x + 1$
$+_A = \lambda xy.x + y$
$\ldots$

$f : X \to |A|$
$f(i) = 10, f(j) = 5, f(k) = 42$
$f(s_1) = "lambda", f(s_2) = "conf"$

$f^{\#} : T_{\Sigma}(X) \to A$
$f^{\#}(toString(succ(j) + succ(0)) + +s_1) = "7lambda"$
$f^{\#}(s_2 + +toString(k) + +s_2) = "lambda42conf"$

MEANING OF FREE

- **Free to interpret in any way**
  - no constraints
- **Free of additional structure**
  - only what's absolutely necessary
- *No junk, no confusion*

ALGEBRAS WITH EQUATIONS

- Let's add constraints on interpretations of a signature
  - that is, on the algebras
- Equations:

$$\Sigma\text{-equation: } \forall X.t = t'$$
$$X\text{: variables, with sorts}$$
$$t, t' \in T_\Sigma(X)\text{: terms with these variables}$$

- And let's focus only algebras satisfying a set Φ of equations

ALGEBRAS WITH EQUATIONS

$\Sigma = (S, \Omega)$, $S = \{v\}$ and $\Omega = \{1, *\}$

$$
\begin{aligned}
\Phi: \quad & \forall\{x, y, z\} x * (y * z) = (x * y) * z \\
& \forall\{x\} x * 1 = x \\
& \forall\{x\} 1 * x = x
\end{aligned}
$$

TERM ALGEBRAS WITH EQUATIONS

- $T_\Sigma(X)$ usually doesn't satisfy the equations
- Some terms need to be "combined"
- Equivalence relation generated by Φ: $\equiv_\Phi$
- Instead of terms, **sets of terms**
  - equivalent wrt equations from Φ

TERM ALGEBRAS WITH EQUATIONS

$\Sigma = (S, \Omega),\ S = \{v\}$ and $\Omega = \{1, *\}$

$$\Phi : \forall\{x, y, z\} x * (y * z) = (x * y) * z$$
$$\forall\{x\} x * 1 = x$$
$$\forall\{x\} 1 * x = x$$

$$T_\Sigma(X)/\equiv_\Phi : \{1\}$$
$$\{x, x * 1, 1 * x\}$$
$$\{x * (y * z), (x * y) * z, (1 * x) * (y * z), 1 * (x * (y * z)), ...\}$$

FREE ALGEBRAS WITH EQUATIONS

Theorem 1 *For any variable set X and equations Φ, $T_\Sigma(X)/\equiv_\Phi$ is free in the class of algebras satisfying Φ*

- Homomorphism to any other algebra
  - how to interpret a set of terms?
  - interpret **any** of them

FREE ALGEBRAS WITH EQUATIONS

$\Sigma = (S, \Omega),\ S = \{v\}$ and $\Omega = \{1, *\}$

$$\Phi : \forall\{x, y, z\} x * (y * z) = (x * y) * z$$
$$\forall\{x\} x * 1 = x$$
$$\forall\{x\} 1 * x = x$$

$$T_\Sigma(X)/\equiv_\Phi : \{1\}$$
$$\{x, x * 1, 1 * x\}$$
$$\{x * (y * z), (x * y) * z, (1 * x) * (y * z), 1 * (x * (y * z)), ...\}$$

- Is there a simpler representation?

$[]$
$[x]$
$[x, y, z]$

FREE RECAP

- Algebraic signature: $\Sigma = (S, \Omega)$
- All possible interpretations: algebras
- For any variable set X
- The term algebra $T_\Sigma(X)$ is **free**
  - any interpretation of the variables $f : X \to |A|$
  - determines an interpretation of any term $f^\# : T_\Sigma(X) \to A$

- A general construction

MODELLING SEQUENTIAL PROGRAMS: MONADS

- A sequential program can:
  - return a value (`pure`)
  - compute what to do next basing on previous result (`flatMap`)
- People decided to call an object with such operations a *Monad*
- Hence, we'll use *Monads* to represent programs as data
  - + sanity laws

FREE MONAD

- *Signature* ~ `pure` + `flatMap`
- *Variables* ~ operations (our `DSL`)
- *Free Monad* ~ terms built out of `pure`, `flatMap`, our `DSL`
  - modulo monad laws!
  - e.g. `flatMap(pure(x), f) = f(x)`

Interpretation of the DSL determines the interpretation of the whole program

FREE IN CATS/SCALAZ

```
trait Free[F[_], A]

object Free {
  case class Pure[F[_], A](a: A)          extends Free[F, A]
  case class Suspend[F[_], A](a: F[A])    extends Free[F, A]
  case class FlatMapped[F[_], B, C](
      c: Free[F, C], f: C => Free[F, B]) extends Free[F, B]
}
```

FREE IN HASKELL

```
data Free f r = Free (f (Free f r)) | Pure r
```

```
trait Free[F[_], A]

object Free {
  case class Pure[F[_], A](a: A)                 extends Free[F, A]
  case class Join[F[_], A](f: F[Free[F, A]]) extends Free[S, A]
}
```

f/F[_] *must be a functor!*

SUMMING UP

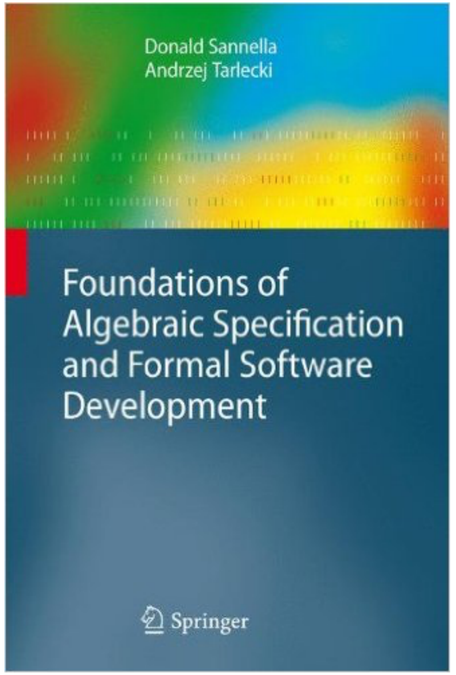
- Direct construction of free algebras
- Hand-wavy construction of free monad
- Free
  - free to interpret in any way
  - free of constraints
  - *no junk, no confusion*
- Free in Haskell is the same free as in Scala

ABOUT ME

- Software Engineer, co-founder @  SOFTWAREMILL
- Custom software development; `Scala/Java/Kafka/Cassandra/...`
- Open-source: `Quicklens, MacWire, ElasticMQ, ScalaClippy`, ...
- Long time ago: student of Category Theory

FURTHER READING

- “Foundations of Algebraic Specification and Formal Software Development” by Donald Sannella and Andrzej Tarlecki
- The Internet



THANK YOU!