

Domain Specific Languages and Towers of Abstraction

Gershon Bazerman

Part I:

Numbers

Numbers

- **Numbers** have **Digits**
- Digits are 0,1,2,3,4,5,6,7,8,9

Here are Some Numbers

- 3
- 34
- 25

Numbers with only digits we call **Naturals**

Here are some more numbers

- 23.5
- 18.21
- 0.9
- 5.0

Numbers with digits and a dot and more digits
we call **Reals**

Reals \rightarrow Naturals

- Take away the dot and subsequent digits. If those digits are nonzero, add one.
- Aka, “**ceiling**”
- Alternately, “**forget**”

Naturals \rightarrow Reals

- Stick on a .o
- Call this “lift” or “**free**”.

Some Numbers are Bigger than Other Numbers

- A **Preorder** has \leq
- $x \leq y$ and $y \leq z$ gives $x \leq z$
- But “not $x \leq y$ ” does **not** give “ $y \leq x$ ”
- Since numbers have an order, they have a preorder

Here's Something Fun

$x., y. \in \text{Reals}$

$x, y \in \text{Naturals}$

$\text{lift } x \leq y. \iff x \leq \text{ceiling } y.$

$\text{lift } x \leq y. \iff x \leq \text{forget } y.$

$\leq \text{ on Reals} \iff \leq \text{ on Naturals}$

$4.0 \leq 4.5 \iff 4 \leq 5$

This Relation on Preordered Sets is a
Galois Connection

This Galois Connection Respects Semirings

- $\text{lift } (+) :: (\text{Nat}, \text{Nat}) \rightarrow \text{Nat} ===$
 $(+) :: (\text{Real}, \text{Real}) \rightarrow \text{Real}$
- $\text{lift } (*) :: (\text{Nat}, \text{Nat}) \rightarrow \text{Nat} ===$
 $(*) :: (\text{Real}, \text{Real}) \rightarrow \text{Real}$
- $\text{forget } (+) :: (\text{Real}, \text{Real}) \rightarrow \text{Real} ===$
 $(+) :: (\text{Nat}, \text{Nat}) \rightarrow \text{Nat}$
- $\text{forget } (*) :: (\text{Real}, \text{Real}) \rightarrow \text{Real} ===$
 $(*) :: (\text{Nat}, \text{Nat}) \rightarrow \text{Nat}$

- $1.0 + 4.0 \leq 5.5 \Leftrightarrow 1 + 4 \leq 6$

- $1.0 + 4.0 \leq 2.9 + 2.9 \Leftrightarrow 1 + 4 \leq 3 + 3$

- $5.0 \leq 1.9 * 2.9 \Leftrightarrow 5 \leq 2 * 3$

Now Log and Exp which respect Semigroups

- $\text{forget}(x) = \ln(x)$

- $\text{lift}(x) = \exp(x)$

- $\text{forget}(*) = (+)$

- $\text{lift}(+) = (*)$

$$\exp x \leq y \quad \Leftrightarrow \quad x \leq \ln y.$$

Pop Quiz

$$98 * 34 < 123456 ?$$

- How many people know the answer to this?
- How many people performed the multiplication to learn the answer?

Knowing beyond Calculating

- We can answer some questions without computing an entire result.
- The formalization of this knowing beyond calculating is an **adjunction**.

- $\text{lift } x \leq y. \Leftrightarrow x \leq \text{forget } y.$
- $\text{Real} \rightarrow \text{Log}(\text{R}) \rightarrow \text{Ceil}(\text{Log}(\text{R}))$
- $98 \rightarrow \text{Log}(98) \rightarrow \text{Ceil}(\text{Log}(98)) = 2$
- $34 \rightarrow \text{Log}(34) \rightarrow \text{Ceil}(\text{Log}(34)) = 2$
- $2 + 2 \leq 4$
- $100 * 100 \leq 10000$
- $10000 < 123456$
- Elementary School Shortcuts are Adjunctions
- Arithmetic Equations are a great
Domain Specific Language for Numbers.

Categories

- Have objects
- Have arrows (morphisms)
- Have conditions (identity and composition)

Functors

- Take object to objects
- Take arrows to arrows
- Preserve identity, Preserve composition

Adjoint Functors

- $F \dashv G$
- $F : D \rightarrow C$
- $G : C \rightarrow D$
- $F \rightarrow F \circ (G \circ F) \rightarrow (F \circ G) \circ F \rightarrow F$
- $G \rightarrow (G \circ F) \circ G \rightarrow G \circ (F \circ G) \rightarrow G$

Or this:

$$\begin{array}{ccc}
 \mathrm{Hom}_{\mathcal{C}}(FY, X) & \xlongequal{\Phi_{Y, X}} & \mathrm{Hom}_{\mathcal{D}}(Y, GX) \\
 \mathrm{Hom}(Fg, f) \downarrow & & \downarrow \mathrm{Hom}(g, Gf) \\
 \mathrm{Hom}_{\mathcal{C}}(FY', X') & \xlongequal{\Phi_{Y', X'}} & \mathrm{Hom}_{\mathcal{D}}(Y', GX')
 \end{array}$$

Galois Connections

- Partially Ordered Sets as a Category
- morphism between x and $y \iff x \leq y$
- Lift \dashv Ceiling
- Exp \dashv Log

Intuitions from Galois Connections

- Functors have a “forgetful” and “free” side
- The “free” side is the Left one.
- The forgetful side tends to smush things.
- It smushes all in one direction.
- The free side does the “one obvious” thing.
- Every right adjoint has only one left (upto iso)
- Vice versa
- Adjunctions compose to form new Adjunctions.

Part II:

Adjunctions and Programming Languages

Every Language has a Theory

- Language = Things you can Say
- Theory = What you can say **about** those things.

Some languages have **bad** theories

```
<?xml version="1.0" encoding="UTF-8"?>
<modification>
  <id>After ABC, add 123 only if XYZ not in file</id>
  <version>1.0</version>
  <vqmver>2.X</vqmver>
  <author>xxx</author>
  <file name="path/to/myfile.php">
    <operation info="After ABC, add 123 if XYZ not in file">
      <ignoreif><![CDATA[
        XYZ
      ]]></ignoreif>
      <search position="after"><![CDATA[
        $var = 'ABC';
      ]]></search>
      <add><![CDATA[
        $var = '123';
      ]]></add>
    </operation>
  </file>
</modification>
```


- Things are similar, therefore equational reasoning is possible.
- Things are different, therefore equational reasoning is necessary.
- Designing a language is balancing between the two — allowing things that are sufficiently different, but no more!
- We want languages open to multiple, nontrivial models.
- This is a job for adjunctions

The Adjunction between Syntax and Semantics

Syntax \dashv Semantics

Model : Sentence \rightarrow a

Theory : Set Sentence

Syntax : Models \rightarrow Theory

Semantics : Theory \rightarrow Models

Syntax is also called “structure”

Here's a Theory

```
data Expr = Sum Expr Expr
          | Product Expr Expr
          | Val Double
```

```
data ExprF a = SumF a a
              | ProductF a a
              | ValF Double
instance Functor ExprF where ...
newtype Fix f = Fix (f (Fix f))
```

```
newtype Mu f =  
    Mu {runMu :: forall a. (f a -> a) -> a }  
  
fixToMu :: Functor f => Fix f -> Mu f  
fixToMu (Fix expr) =  
    Mu $ \ f -> f . fmap (($f) . runMu . fixToMu) $ expr
```

```
type Sentence f = Fix f
```

```
type Model f a = f a -> a
```

```
runInterp :: Functor f => Model f a -> Sentence f -> a  
runInterp i = \e -> runMu (fixToMu e) i
```

```
interpExp :: Model ExprF Double  
interpExp (SumF x y) = x + y  
interpExp (ProductF x y) = x * y  
interpExp (ValF d) = d
```

```
-- runInterp interpExp simpleExpr = 3
```

Models have Adjoints!

```
type Model f a = f a -> a
type CoModel f a = a -> f a
adjModel :: (f a -> a) -> (a -> f a)
```

adjModel finds the minimal “f a” that yields a.

Models yield Adjoints!

```
runInterp      :: Model f a -> Sentence f -> a  
findSentence :: Model f a -> a -> Sentence f
```

`runInterp` finds the unique A given by the sentence.

`findSentence` = find the minimal sentence that yields an A .

`findSentence m = Fix . adjModel m`

`findSentence . runInterp ===== Normalization by Evaluation`

Models have Validity

```
type Test a = a -> Bool

-- we can take: Model f a → Model f Bool

semantics ::
  Test a -> Set (Sentence f) -> Set (Model f a)

syntax ::
  Test a -> Set (Model f a) -> Set (Sentence f)
```

- More models = fewer theories
- More theories = fewer models
- Galois Connection

Another Example

- Language is polynomial expressions in 3 variables
- Semantic Domain is Reals
- Models are triples representing substitutions
- Validity judgement is equality to zero
- More formulae to satisfy = Fewer assignments work
- More assignments = Fewer formulae are satisfied by them

Adjoint Properties

- $\text{findModels} \circ \text{findTheories} \circ \text{findModels} = \text{findModels}$
- $\text{findTheories} \circ \text{findModels} \circ \text{findTheories} = \text{findTheories}$
- $\text{findTheories} \circ \text{findModels} = \text{closure of the models. If you have the first set you might as well have all the rest.}$
- $\text{findModels} \circ \text{findTheories} = \text{closure of the theory. If you can say these sentences, you might as well say the rest.}$

Morphisms between Theories are Natural Transformations

```
type Natural f g = forall a. f a -> g a
```

```
data Expr2F a = Sum2F a a
               | Product2F a a
               | Val2F Integer
```

```
trans :: Natural ExprF Expr2F
```

```
trans (SumF x y) = Sum2F x y
```

```
trans (ProductF x y) = Product2F x y
```

```
trans (ValF d) = Val2F . ceiling . log $ d
```

```
transToModel ::  
    Natural f g -> Model f (Sentence g)  
transToModel eta = Fix . eta  
  
morphSentence :: Functor f =>  
    Natural f g -> Sentence f -> Sentence g  
morphSentence eta = runInterp  
                    (transToModel eta)
```

Every theory is someone else's semantic domain.

Chains of transformation give rise to chains of adjunctions
give rise to towers of semantics.

(aside) What about Effects?

- Effects break referential transparency
- Names (let/lambda), Mutation, Exceptions
- Capture Effects in your Semantic Domain
- $\text{Monad } m \Rightarrow \text{Model } f (m \ a)$

What is the correct Semantic Domain for Programs?

The problem is in capturing recursive definitions.

The answer to this question leads us to
Denotational Semantics

(and a whole other talk).

Part II:

Applications

So?

- Don't start with Theories (syntax)
- Start with Semantic Domains (combinators)
- Write theories that match your domains
- Layer theories on theories, with each model disallowing more sentences, and providing more rules
- Include an AST -- leave yourself open to multiple interpretations

adf-dfa

- Applicative Combinators
- Haskell DFA Combinators
- Monadic Transition Collections
- Transition Collections
- LLVM AST
- LLVM Bytecode
- Assembly

adf-dfa

- String
- NFA
- Transition Collections
- LLVM AST
- LLVM Bytecode
- Assembly

adf-dfa

- Applicative Combinators
- Haskell DFA Combinators
- Monadic Transition Collections
- Transition Collections
- Direct Interpretation

Abstract Interpretation

- Model $T \ D$
- $T \rightarrow D$
- Model $T' \ D'$ is an abstraction of Model $T \ D$ when
- $C : D' \rightarrow D \dashv A : D \rightarrow D'$
- $C : T \rightarrow T' \dashv A : T \rightarrow T'$
- $T' \rightarrow T \rightarrow D \leq T' \rightarrow D' \rightarrow D$

Things we may want to do

- Find extremum
- Find datasources/effects
- Check/infer types
- Guarantee “safety”
- Partial evaluation

Languages Computers can
Reason About



Languages People can Reason
About

P.S.

- Every adjunction gives rise to a monad.
- Every monad can be factored into an adjunction.
- Adjunctions are everywhere, once you know what you're looking for.

Further Reading

- Cousot, P. “Constructive Design of a hierarchy of Semantics of a Transition System by Abstract Interpretation,” 2002.
- Hyland and Power. “The Category Theoretic Understanding of Universal Algebra: Lawvere Theories and Monads,” 2007.
- Lawvere, F.W. “Functorial Semantics of Algebraic Theories,” 1963.
- Lawvere, F.W. “Adjointness in Foundations,” 1969.
- Meijer, Fokkinga and Paterson. “Functional Programming with Bananas, Lenses, Envelopes and Barbed Wire.”
- Smith, Peter. “The Galois Connection Between Syntax and Semantics,” 2010.

- $T' \rightarrow T \rightarrow D \leq T' \rightarrow D' \rightarrow D$
- $T \rightarrow T' = \text{Theory ExprF} \rightarrow \text{Theory Expr2F}$
- $T' \rightarrow T = \text{Theory Expr2F} \rightarrow \text{Theory ExprF}$
- $D \rightarrow D' = \text{ceiling} . \log$
- $D' \rightarrow D = \text{exp} . \text{fromIntegral}$
- “count the digits” is an abstract interpretation.

JMacro Panels

- Reactive Combinators ->
- Reactive Semantics ->
- Communication Semantics ->
- Hygienic JavaScript Semantics ->
- Reduced JavaScript AST ->
- String