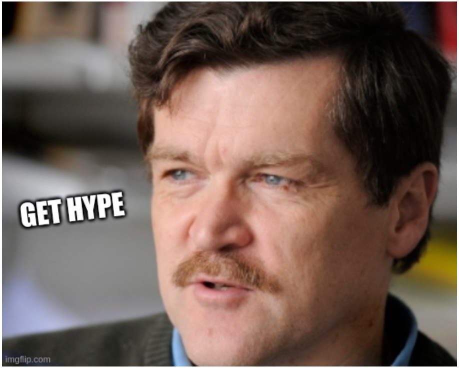
Algebraic Data Types

Hype for Types

September 9, 2020

Outline

- Look at types we already know from a different angle
- Formalize some important new type concepts
- break the universe



Introduction to Counting

bool and order

Notation

Write $|\tau|$ to denote the number of elements in type τ.

```
datatype bool  = false | true
datatype order = LESS | EQUAL | GREATER
```

What size are they?

$$|\textbf{bool}| = 2$$
$$|\textbf{order}| = 3$$

Often, we refer to **bool** as 2 and **order** as 3:

$$\texttt{true} : 2$$
$$\texttt{LESS} : 3$$

Products

Products

Question

What is $|\tau_1 \times \tau_2|$?

$|\tau_1| \times |\tau_2|$ - hence, the notation.

For example,

$$\begin{aligned}
|\textbf{bool} \times \textbf{order}| &= |\textbf{bool}| \times |\textbf{order}| \\
&= 2 \times 3 \\
&= 6
\end{aligned}$$

What do you know!

Theorem: Commutativity of Products

For all τ_1, τ_2:

$$\tau_1 \times \tau_2 \simeq \tau_2 \times \tau_1$$

Theorem: Associativity of Products

For all τ_1, τ_2, τ_3:

$$\tau_1 \times (\tau_2 \times \tau_3) \simeq (\tau_1 \times \tau_2) \times \tau_3$$

Question

How do we know?

Proving Type Isomorphisms

To prove that $\tau \simeq \tau'$, we need a *bijection* between τ and τ'.

We write two (total) functions, $f : \tau \rightarrow \tau'$ and $f' : \tau' \rightarrow \tau$, such that f and f' are *inverses*.

$$\texttt{f' (f x)} \cong \texttt{x}$$

$$\texttt{f (f' x)} \cong \texttt{x}$$

Associativity of Products: Proved!

Let's prove associativity of products:

$$\tau_1 \times (\tau_2 \times \tau_3) \simeq (\tau_1 \times \tau_2) \times \tau_3$$

Need to write:

$$f : \tau_1 \times (\tau_2 \times \tau_3) \to (\tau_1 \times \tau_2) \times \tau_3$$
$$f' : (\tau_1 \times \tau_2) \times \tau_3 \to \tau_1 \times (\tau_2 \times \tau_3)$$

Nice!

$$f = \texttt{fn (a,(b,c)) => ((a,b),c)}$$
$$f' = \texttt{fn ((a,b),c) => (a,(b,c))}$$

Multiplicative Identity?

Follow-Up

Is there an identity element, "1"?

$$\tau \times 1 = \tau$$
$$1 \times \tau = \tau$$

Yes - **unit**!

Theorem

For all types τ:

$$\tau \times \textbf{unit} \simeq \tau$$
$$\textbf{unit} \times \tau \simeq \tau$$

Sums

Increment

Question

Is there such thing as $\tau + 1$?

Answer

Yes! τ **option**.

SOME x	(τ choices)
NONE	(1 choice)

Sums

```
datatype ('a,'b) either = Left of 'a | Right of 'b
```
[1]

$$\frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash \textbf{Left}\ e : \tau_1 + \tau_2}\ (\textsc{Left}) \qquad \frac{\Gamma \vdash e : \tau_2}{\Gamma \vdash \textbf{Right}\ e : \tau_1 + \tau_2}\ (\textsc{Right})$$

$$\frac{\Gamma \vdash e : \tau_1 + \tau_2 \qquad \Gamma, x_1 : \tau_1 \vdash e_1 : \tau \qquad \Gamma, x_2 : \tau_2 \vdash e_2 : \tau}{\Gamma \vdash \textbf{case}\ e\ \textbf{of}\ x_1 \Rightarrow e_1 \mid x_2 \Rightarrow e_2 : \tau}\ (\textsc{Case})$$

And of course...

For all τ_1, τ_2:

$$|\tau_1 + \tau_2| = |\tau_1| + |\tau_2|$$

[1] In the Standard ML Basis, (almost) the `Either` structure!

Options as Sums

```
datatype ('a,'b) either = Left of 'a | Right of 'b
```

Notice:

```
type 'a option = ('a,unit) either
```

We can represent τ **option** as $\tau + \textbf{unit}$.

Example: Distributivity

Claim

For all types A, B, C:

$$(A \times B) + (A \times C) \simeq A \times (B + C)$$

f : `('a * 'b, 'a * 'c) either -> 'a * ('b,'c) either`

f' : `'a * ('b,'c) either -> ('a * 'b, 'a * 'c) either`

$f =$ `fn Left (a,b) => (a,Left b) | Right (a,c) => (a,Right c)`

$f' =$ `fn (a,Left b) => Left (a,b) | (a,Right c) => Right (a,c)`

Practical Application

Code refactoring principle! If both cases store the same data, factor it out.

Zero to Hero

If we can add, what's 0?

We call it **void**, the empty type.[2]

$$\frac{\Gamma \vdash e : \textbf{void}}{\Gamma \vdash \textbf{absurd}(e) : \tau}\ (\textsc{absurd})$$

Implementing via SML Hacking

```
datatype void = Void of void
fun absurd (Void v) = absurd v
```

Notice: `absurd` is total!

[2] Unlike C's `void` type, which is actually **unit**.

void*

Claim

For all types τ:

$$\tau + \textbf{void} \simeq \tau$$

$$\begin{aligned} f &: \texttt{('tau,void) either -> 'tau} \\ f' &: \texttt{'tau -> ('tau,void) either} \end{aligned}$$

$$\begin{aligned} f &= \texttt{fn Left x => x | Right v => absurd v} \\ f' &= \texttt{fn x => Left x} \\ &= \texttt{Left} \end{aligned}$$

Functions

How Many Functions?

How many (total) values are there of type $A \to B$, in terms of $|A|$ and $|B|$?

- How many functions are there of type $2 \to A$? $|A| \times |A|$
- How many functions are there of type $A \to 2$? $2^{|A|}$

Theorem

There are $|B|^{|A|}$ total functions from type A to type B.

Example: Power of a Power

In math, it's true that:

$$(C^B)^A = C^{A \times B}$$

In terms of types, that would mean:

$$A \to (B \to C) \simeq A \times B \to C$$

Yes!

```
f  = Fn.uncurry : ('a -> 'b -> 'c) -> ('a * 'b -> 'c)
f' = Fn.curry   : ('a * 'b -> 'c) -> ('a -> 'b -> 'c)
```

Recursive Types

Lists

```
datatype 'a list = Nil | Cons of 'a * 'a list
```

```
datatype 'a list = Left of unit | Right of 'a * 'a list
```

```
type 'a list = (unit, 'a * 'a list) either
```

$$L(\alpha) \simeq \textbf{unit} + \alpha \times L(\alpha)$$

$$\begin{aligned} L(\alpha) &= 1 + \alpha \times L(\alpha) \\ &= 1 + \alpha \times (1 + \alpha \times L(\alpha)) \\ &= 1 + \alpha + \alpha \times L(\alpha) \\ &= 1 + \alpha + \alpha \times (1 + \alpha \times L(\alpha)) \\ &= 1 + \alpha + \alpha^2 + \alpha^3 + \dots \end{aligned}$$

Binary Trees

```
datatype 'a tree
  = Empty
  | Node of 'a tree * 'a * 'a tree
```

$$
\begin{aligned}
T(\alpha) &\simeq \textbf{unit} + T(\alpha) \times \alpha \times T(\alpha) \\
&\simeq \textbf{unit} + \alpha \times T(\alpha)^2
\end{aligned}
$$

Binary Shrubs

```
datatype 'a shrub
  = Leaf of 'a
  | Node of 'a shrub * 'a shrub
```

$$\begin{aligned} S(\alpha) &\simeq \alpha + S(\alpha) \times S(\alpha) \\ &\simeq \alpha + S(\alpha)^2 \end{aligned}$$

Natural Numbers

How many natural numbers are there?

```
datatype nat = Zero | Succ of nat
```

$$\textbf{nat} = \textbf{unit} + \textbf{nat}$$

$$\textbf{nat} = 1 + 1 + 1 + \cdots = \infty$$

Therefore, we would expect:

$$\infty = 1 + \infty$$

$$\textbf{nat} \simeq \textbf{nat option}$$

```
f  = fn Zero => NONE | Succ n => SOME n
f' = fn NONE => Zero | SOME n => Succ n
```

haha type derivates go brrr

Taking Things Too Far

Question

What is $\frac{d}{d\alpha}\tau(\alpha)$?

Smart Idea

Dismiss the idea outright - this is madness!

Our Plan

$>$:)

>:)

$$\frac{d}{d\alpha}\alpha^3 = 3\alpha^2$$

$$\alpha \times \alpha \times \alpha \qquad \mapsto \qquad 3 \times (\alpha \times \alpha)$$

Conclusion

Differentiating a power "eats" a tuple slot, and tells you which element was removed.

Differentiating a List

Recall from calculus (?) that:

$$a + ar + ar^2 + ar^3 + \cdots = \frac{a}{1-r}$$

We have:[3]

$$L(\alpha) = 1 + \alpha + \alpha^2 + \ldots \stackrel{?}{=} \frac{1}{1-\alpha}$$

$$\begin{aligned} \frac{d}{d\alpha} L(\alpha) &= \frac{d}{d\alpha} \frac{1}{1-\alpha} \\ &= \frac{1}{(1-\alpha)^2} \\ &= \left(\frac{1}{1-\alpha} \right)^2 \\ &= L(\alpha)^2 \end{aligned}$$

[3] What the hype is a negative type?

Tree for Two, and Two for Tree

We said:

$$T(\alpha) = 1 + \alpha T(\alpha)^2$$

Here we go again...

$$\begin{aligned}
\frac{d}{d\alpha} T(\alpha) &= \frac{d}{d\alpha} 1 + \frac{d}{d\alpha} \alpha T(\alpha)^2 \\
&= \alpha \times \frac{d}{d\alpha} T(\alpha)^2 + \frac{d}{d\alpha} \alpha \times T(\alpha)^2 \\
&= 2\alpha T(\alpha) \times \frac{d}{d\alpha} T(\alpha) + T(\alpha)^2 \\
\frac{d}{d\alpha} T(\alpha) &= T(\alpha)^2 \left(\frac{1}{1 - 2\alpha T(\alpha)} \right) \\
&= T(\alpha)^2 L(2\alpha T(\alpha))
\end{aligned}$$

Holey Cow!

$$\frac{d}{d\alpha}\alpha^3 = 3\alpha^2 \qquad \text{"punctured" tuple}$$

$$\frac{d}{d\alpha}L(\alpha) = L(\alpha)^2 \qquad \text{list zipper}$$

$$\frac{d}{d\alpha}T(\alpha) = T(\alpha)^2 L(2\alpha T(\alpha)) \qquad \text{tree zipper}$$

Theorem

The Derivative of a Regular Type is its Type of One-Hole Contexts.[a]

---

[a] http://strictlypositive.org/diff.pdf

Conclusion

Conclusion

- Figured out the sizes of various types
- Generalized our type theory to include *sum types* (and **void**)
- Considered *recursive types*[4]
- Invented a type-level hole punch

[4] More on that later...