Speaking in Types

How I Learned to Stop Worrying and Love the Type System

TJ Usiyan - @griotspeak

What is a Type?

What is a Type?

The description of a portion of our system's domain.

What is a Type?

Types are how we describe the data that our system works with.

In Objective-C, Compiler puts trust in you. In Swift, you put trust in compiler
– Yakko Smirnoff

I'm Sorry Dave. I'm Afraid I can't do that.

Value vs Reference

@autoclosure on recursive types

"An Either by any other name would compile as sweet."

– Gilliam Shakespeare

```
enum Either<T, U>  {
    case Left(T)
    case Right(U)
}

enum Result<T>  {
    case Error(NSError)
    case Success(T)
}
```

Examining Types in Xcode

- Option click
- Look at the generated header

On Semantics

Demo

Make Illegal States Unrepresentable

– Yaron Minsky

Enums Are Delightful.

```
enum FileLoadingResult {
    case .Success(NSData)
    case .LoadError(NSError)
    case .ProgrammerError(String)
}
```

Excise the jibber jabber from your types

– TJ

Optionals Are Quite Nice as Well.

```
typedef NS_ENUM(NSInteger, YouCanGetWith) {
    CardinalDirectionNull = 0,
    YouCanGetWithThis,
    YouCanGetWithThat};
```

vs

```
enum YouCanGetWith {
    case This, That
}
```

Putting the 'Algebra' in Algebraic Data Types

Algebra of Data Types - Zero Type

Void

```swift
/// The empty tuple type.
///
/// This is the default return type of
/// functions for which no explicit
/// return type is specified.
typealias Void = ()
```

Algebra of Data Types - Sum Types

Enums

```
`enum Either<T, U> {
 case .Left(T)
 case .Right(U)
}

let foo:Either<Bool, Void>
`
```

Algebra of Data Types - Product Types

Tuples

```
let bar:(Bool, Void) // 2 possible values

let baz:(Bool, Bool)
```

Functions Types Need Love Too.

– Samantha Wolf

Functions are Maps!

```
func myRandom() -> Int
func consumeNumber(Int) // (Int) -> ()
func sortNums([Int]) -> [Int]
```

Functions are Maps!

Parametric Polymorphism

```
struct Array<T>

class Thing<T>

func foo<T>

protocol BazableType {
    typealias T
}
```

Parametric Polymorphism

- Structs
- Classes
- Prototypes
  - Structs
  - Classes
- Functions

Trait Polymorphism

```
struct Array<T:Equatable>

class Thing<T:Equatable>

func foo<T:Equatable>
```

Trait Polymorphism

- Structs
- Classes
- Functions

Generic Function Signatures

```
<T>(T) -> ()
```

Generic Function Signatures

```
<T>([T]) -> [T]
```

Generic Function Signatures

```
<T: Comparable>([T]) -> [T]
```

Generic Function Signatures

```
<T, U>(T) -> U
```

Generic Function Signatures

```
<T>(T, T) -> T
```

Generfic Function Signatures

```
<T,U,V>((U -> V), (T -> U)) -> (T -> V)
```

Names Are Part of a Type

```
func stride<T : Strideable>(from start: T, to end: T, by stride: T.Stride) -> StrideTo<T>
func stride<T : Strideable>(from start: T, through end: T, by stride: T.Stride) -> StrideThrough<T>
```

Names Are Part of a Type

… start: T, **to** end: …

… start: T, **through** end: …

Function Overloading

Function Overloading

```
+(T, T) -> T
```

Function Overloading

```
func +(lhs: Float, rhs: Float) -> Float
func +<T>(lhs: Int, rhs: UnsafePointer<T>) -> UnsafePointer<T>
func +<T>(lhs: UnsafePointer<T>, rhs: Int) -> UnsafePointer<T>
func +(lhs: Int, rhs: Int) -> Int
func +(lhs: UInt, rhs: UInt) -> UInt
func +(lhs: Int64, rhs: Int64) -> Int64
func +(lhs: UInt64, rhs: UInt64) -> UInt64
func +<T>(lhs: Int, rhs: UnsafeMutablePointer<T>) -> UnsafeMutablePointer<T>
func +<T>(lhs: UnsafeMutablePointer<T>, rhs: Int) -> UnsafeMutablePointer<T>
func +(lhs: Int32, rhs: Int32) -> Int32
func +(lhs: UInt32, rhs: UInt32) -> UInt32
func +(lhs: Int16, rhs: Int16) -> Int16
func +(lhs: UInt16, rhs: UInt16) -> UInt16
func +(lhs: Int8, rhs: Int8) -> Int8
func +(lhs: UInt8, rhs: UInt8) -> UInt8
func +(lhs: Double, rhs: Double) -> Double
func +(lhs: String, rhs: String) -> String
```

Optional Initializers

...when illegal state is completely worth it

```
struct ScaleDegree {
    init(_ diatonicValue:DiatonicValue, _ chromaticValue:ChromaticValue)
    init(_ accidental:Accidental, _ diatonicValue:DiatonicValue)
    init?(_ quality:Interval.Quality, _ diatonic:DiatonicValue)
}

let scaleDegree = ScaleDegree(.Perfect, .Fifth)!
```

Higher Kinded Types

Higher Kinded Types

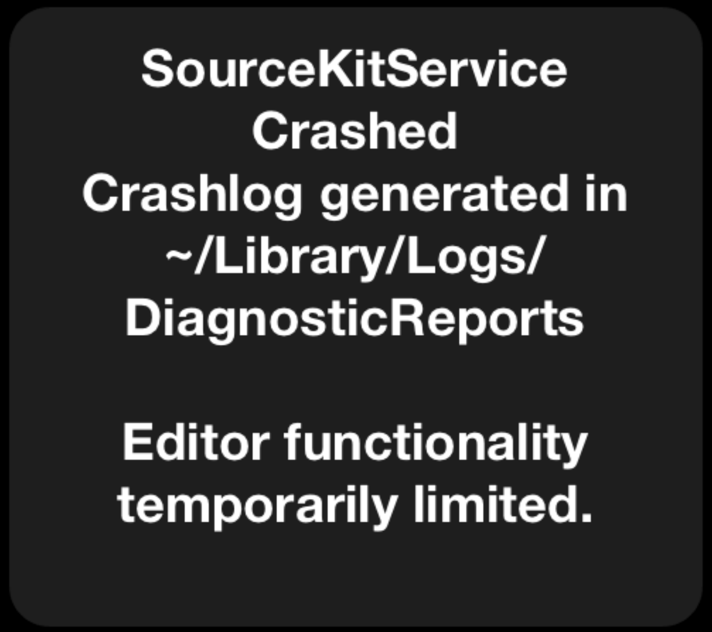
```
func ==<T : Equatable>(lhs: T?, rhs: T?) -> Bool
/// Returns true if these arrays contain the same elements.
func ==<T : Equatable>(lhs: [T], rhs: [T]) -> Bool
/// Returns true if these arrays contain the same elements.
func ==<T : Equatable>(lhs: Slice<T>, rhs: Slice<T>) -> Bool
/// Returns true if these arrays contain the same elements.
func ==<T : Equatable>(lhs: ContiguousArray<T>, rhs: ContiguousArray<T>) -> Bool
```

Higher Kinded Types

How can we state that a container holding an Equatable type is Equatable?

Higher Kinded Types

~~How can we state that a container holding an Equatable type is Equatable?~~



Higher Kinded Types

We **must** repeat ourselves. This fact is unfortunate.

- The type system is our friend
- Types can hold more meaning than in Obj-C.

Sources

- Effective ML [1]
- Algebra of Data Types [^2]

---

[1] https://vimeo.com/14313378

[^2]:http://chris-taylor.github.io/blog/2013/02/10/the-algebra-of-algebraic-data-types/